Universidad
Carlos III de Madrid

TRABAJO FIN DE GRADO

Título: Diseño y desarrollo de un monitor para tráfico multimedia en un entorno residencial

Autor: Enol Cordero Fernández
Titulación: Grado en Ingeniería de Sistemas Audiovisuales
Tutor: Francisco Varela Pintor
Fecha: 24/09/2013

Agradecimientos

A mi familia, especialmente a mis padres, por permitirme empezar esta aventura lejos de casa, por apoyarme siempre que lo he necesitado, y cuando no lo he necesitado también. Sin ellos sería imposible haber llegado hasta aquí.

A Sofía, por su cariño, su apoyo, su protección y sus consejos. Por haber hecho que estos cuatro años lejos de casa fuesen un poco menos duros por tenerla a ella cerca.

A mi hermano, Adrián, porque aunque me cabree con él cuando me dice las cosas, sé que lo hace por mi bien.

A Pablo, por esos casi cuatro años compartiendo clase, trabajos y sobre todo amistad. Todo habría sido mucho más difícil sin él a mi lado.

A mi madre y a Sofía una vez más, por leerse toda la memoria sin saber ni por dónde empezar. Vuestras correcciones y vuestros consejos han sido básicos para mejorar el estado final de este trabajo.

A todos mis compañeros, que han hecho mucho más agradable el paso por la Universidad, y que siempre han estado ahí

A todos los profesores con los que he tenido la suerte de coincidir en estos cuatro años, que aunque directamente no tengan nada que ver con este proyecto, han sido básicos en mi formación y en mi crecimiento, que me han permitido llegar hasta aquí. De todos he aprendido algo.

A Alberto Gordillo, por el trabajo con las máquinas virtuales y solucionar rápidamente todos los problemas surgidos. Sin su dedicación habría sido imposible que hoy estuviese escribiendo estas líneas.

Y por supuesto, a Francisco Valera, Paco, por ofrecerme la oportunidad de trabajar con él, un proyecto interesante, guiarme a través de él y ayudarme con los problemas que pudiesen surgir. Me ha ayudado a crecer como estudiante, como futuro ingeniero y como persona.

De corazón…Gracias a todos.

Resumen

El siguiente TFG se basa en el desarrollo de un programa en Java que modifique paquetes SIP [1] (Session Initiation Protocol) que contengan solicitudes INVITE con el fin de mejorar las comunicaciones multimedia, sobre todo las videoconferencias. Esta modificación consistirá en seleccionar los codecs adecuados en función del ancho de banda disponible en la red de acceso del usuario.

El trabajo está dirigido para su instalación en una Whitebox de Samknows[2], enmarcándose dentro del proyecto que la empresa está desarrollando junto con la Comisión Europea.

La solución propuesta se desarrollará en tres partes:

1. **Recepción del paquete:** se desviará el paquete hacia un puerto determinado, donde nuestro programa tendrá una pila escuchando las solicitudes que lleguen y su tipo.
2. **Obtención de la información multimedia:** obtendremos la información multimedia de la carga SDP [3] (Session Description Protocol) y la compararemos con la que nos daría una buena calidad en función del ancho de banda disponible en ese momento, quedándonos con los codecs comunes a ambos.
3. **Modificación del mensaje:** una vez sepamos los codecs comunes, modificaremos la carga SDP y posteriormente el mensaje, para enviarlo modificado al servidor.

Esta modificación se realiza de manera transparente al usuario y en tiempo real, con el fin de obtener un mejor rendimiento y experiencia de usuario en las comunicaciones multimedia, sin que esto suponga una penalización en términos de instalación del software, utilización de éste y tiempo de espera para iniciar las comunicaciones.

Palabras clave: multimedia, videoconferencia, SIP, SDP, códec, paquete, IP, ancho de banda.

Summary

The next Bachelor Thesis is based on the development of a Java program that modifies SIP [1] (Session Initiation Protocol) packets which contain INVITE request with the goal of improving multimedia communications, mainly videoconferences. This modification will consist on selecting the right codecs depending on the available bandwidth at the user access network.

The job is focused for its installation in a Smaknows Whitebox [2], framing itself inside the project the company is developing with the European Commission (EC).

The proposed solution will be developed in three steps:

1. **Packet reception:** the packet will be redirected towards a well-known port, where our program will have a stack which is listening to the arrived requests and its type.
2. **Getting the multimedia information:** we will get the multimedia information from the SDP [3] (Session Description Protocol) load and we will compare it with the one that would give us good quality according to the available bandwidth at this moment, choosing the common codecs.
3. **Message modification:** when we know de common codecs, we will modify the SDP load and the message with it, and we will send de modified packet to the server.

This modification is done transparently for the user and in real-time, with the purpose of getting a better performance and user experience in the multimedia communications, but don't supposing a penalization in terms of software installation, use and wait time to start the communications

Keywords: multimedia, videoconference, SIP, SDP, codec, packet, IP, bandwidth.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Motivación y objetivos

1.1 Motivación

En ocasiones, las comunicaciones multimedia no son todo lo fluidas que deberían ser; se producen cortes, saltos, pixelado y muchos problemas más que resultan muy incómodos para el usuario. Esto, además de molesto, puede ser un gran inconveniente en videoconferencias o llamadas de VoIP con objetivo profesional, imposibilitando incluso cualquier tipo de conversación, presentación de un proyecto o negociación.

A pesar de que cada vez más programas como Skype tratan de mitigar estos problemas con diferentes técnicas, como reducción del bitrate de salida en función del ancho de banda o redundancia en los paquetes, en muchas ocasiones esto no es suficiente, pues quizás otro códec sería necesario, incluso uno que no esté disponible en ese programa.

Así, podemos comprobar, como se desprende de [4][5], que incluso en condiciones en las que el ancho de banda es suficiente, el MOS (Mean Opinion Score), suele estar en torno a 4 sobre 5, lo que responde a una calidad media-alta, pero en ningún caso nos colocamos en una situación de calidad alta-muy alta, por lo que en cuanto el ancho de banda disponible se reduzca ligeramente, estaremos en un MOS medio-bajo. También del estudio [6] se desprende que, al reducirse el bitrate, aumenta la pérdida de paquetes, y por tanto disminuye la calidad percibida por el usuario. Además, al añadir una mayor redundancia en los paquetes para tratar de mitigar las pérdidas, estos aumentan su tamaño y por tanto la cantidad de datos enviados, lo que puede provocar que la red sufra una mayor saturación.

En este proyecto vamos a tratar de solucionar estos problemas, de manera que nuestras comunicaciones multimedia sean mucho más fluidas y útiles, sin cortes que compliquen nuestras relaciones tanto personales como profesionales, obteniendo una experiencia de usuario mejor.

1.2 Objetivos

El objetivo principal de este proyecto es evitar los problemas mencionados anteriormente, de manera totalmente transparente al usuario, que realizará sus llamadas, videoconferencias, etc. de la misma forma que lo había hecho hasta entonces. Para ello, trataremos de seleccionar los mejores codecs en función del ancho de banda en cada momento, con el fin de conseguir una experiencia de usuario superior.

Para conseguir este objetivo, desarrollaremos un programa en Java que, al recibir una solicitud INVITE, modifique su carga SDP incluyendo únicamente los codecs que se ajusten al ancho de banda disponible en la red del usuario.

1.3 Estructura de la memoria

Vamos a presentar un pequeño resumen de cada uno de uno de los capítulos con el objetivo de que se comprenda mejor el trabajo y su estructura.

En primer lugar, en el estado del arte se hará un estudio teórico sobre los diferentes protocolos y tecnologías que se utilizarán en el desarrollo del trabajo.

Tras la parte teórica, en el capítulo tres nos dedicaremos a explicar el diseño y desarrollo de la solución. En la parte de diseño valoraremos las diferentes alternativas valoradas y por qué elegimos las soluciones que finalmente se han adoptado; estas soluciones se explican en detalle en el apartado de desarrollo.

En el capítulo cuatro se detallarán las pruebas realizadas y se comentarán los resultados obtenidos, valorándolos dentro de los diferentes contextos y sacando las conclusiones necesarias.

Posteriormente, en el capítulo cinco, detallaremos la planificación seguida en el desarrollo del proyecto y el presupuesto necesario para completarlo. Además, se explicará el entorno socioeconómico para el que está enfocado el proyecto.

Por último, en el capítulo seis, se expondrán las conclusiones que se obtienen del trabajo realizado, y se plantean diferentes ideas para continuar con el trabajo en el futuro, de manera que se ofrezca una visión de cómo se podría completar y mejorar el proyecto y obtener mejores resultados.

Capítulo 2

Estado del arte

2.1 Introducción

En este apartado se describirá el problema planteado y las soluciones tecnológicas elegidas para su desarrollo, así como el porqué de estas decisiones. Se cubrirá el apartado teórico del proyecto.

2.2 Planteamiento del problema

Las comunicaciones multimedia no suelen tener en cuenta las condiciones de la red en la que se van a efectuar para determinar qué calidad usar en ellas. En algunos casos, solo existe una disponible; en otros, existen varias y el servidor elige una por diferentes criterios (mayor calidad, menor tasa, al azar…). Esto puede ocasionar una elección no óptima de los codecs de audio y vídeo a utilizar, y por tanto a una experiencia de usuario pobre.

En la actualidad, algunas plataformas han introducido diferentes calidades en sus vídeos online (por ejemplo, Youtube o las televisiones online), e incluso un modo automático que se adapta a los recursos en cada momento, aunque éste puede llegar a resultar incómodo debido a su alta sensibilidad y a los cortes que se producen al cambiar de una a otra.

En cambio, en el apartado de las comunicaciones bilaterales, como por ejemplo las videoconferencias o las llamadas de VoIP, estas mejoras aún no están incorporadas o existen tantas diferentes posibilidades implementadas en los programas, por lo que la sesión puede no utilizar la solución óptima para que la experiencia de usuario sea todo lo buena que cabría esperar.

En este proyecto vamos a desarrollar un programa en Java que elija, entre los diferentes codecs, el óptimo en función del ancho de banda de acceso disponible al inicio de la sesión. Para ello, utilizaremos el protocolo SIP para el establecimiento de la sesión y RTP/RTCP [7] [8] como protocolo para la comunicación multimedia.

Hemos elegido SIP como protocolo para la negociación de la sesión por ser el aceptado por los estándares, usado de manera universal y definido claramente en la RFC 3261, junto a la RFC 4566 para la especificación de la carga SDP. En cuanto a RTP/RTCP como protocolo para la comunicación multimedia, lo hemos elegido por la gran cantidad de codecs tanto de audio como de vídeo que ofrece, su gran integración con SIP, la información que aportan y la gran cantidad de documentación existente, recogiendo el estándar en las RFC 3550 y 3551.

El cliente SIP realiza una invitación al servidor (u otro cliente, aunque por comodidad hablaremos siempre de servidor) en el que se indican, dentro de la carga SDP, los codecs de audio y vídeo que tiene disponibles y/o que desea utilizar, de manera que el servidor establece la sesión eligiendo uno de ellos, sin ningún criterio específico, por lo que la selección puede no ser la adecuada. Lo que vamos a hacer, es extraer este paquete de invitación de la red, de manera que, sabiendo el ancho de banda de acceso disponible y los codecs que se pueden utilizar, eliminemos aquellos que tienen unas necesidades superiores a las disponibles, para después reenviarlo al servidor. Así, cuando éste reciba la solicitud, cualquiera de las opciones que elija nos proporcionará una comunicación sin cortes.

Para conocer el ancho de banda disponible utilizaremos una *Whitebox* de SamKnows. Mediante pruebas periódicas con diferentes servidores propios y externos calcula el ancho de banda de acceso que tenemos, y va actualizándolo en un documento de texto, de donde nosotros lo obtendremos. Este dispositivo se enmarca dentro de un proyecto de la Comisión Europea, lo que hace aún mayor la importancia del trabajo.

Esta solución puede significar una disminución de la calidad de vídeo y audio, pero evitará efectos desagradables como los cortes o el *blocking* en la comunicación. Otro problema existente es que seguimos sin poder saber qué códec elegirá, por lo que la opción elegida puede seguir sin ser la óptima y proporcionarnos una calidad inferior a la que sería posible, pero esto ya es un problema endémico de SIP.

2.3 Análisis del estado del arte

Como ya se ha comentado, utilizaremos los protocolos SIP, SDP y RTP para las comunicaciones multimedia; Java como lenguaje de programación y una Whitebox como hardware específico.

Elegimos SIP como protocolo para la negociación por ser el estándar en este sentido, siempre acompañado de SDP para la descripción de la carga multimedia. En cuanto a RTP, era la opción que mejor se adaptaba a nuestro trabajo de las que integra SIP, al ofrecer diferentes tipos de codecs y un gran número de datos suplementarios de gran utilidad.

En cuanto a Java como lenguaje de programación, decidimos utilizarlo después de comentar el problema con algunos profesores especialistas en el trabajo con SIP/SDP. Éstos nos recomendaron utilizarlo dado que ya existían una gran cantidad de bibliotecas y soporte para trabajar con estos protocolos, por lo que las opciones de desarrollo aumentaban. En este caso, la primera opción barajada había sido C, pues de esta manera se podía trabajar a bajo nivel, directamente en el Kernel y sin necesidad de enviar nuestro paquete a nivel de aplicación, pero la falta de soporte y librerías desarrolladas nos hicieron decantarnos por Java, que nos ofrecía un mayor número de posibilidades.

La Whitebox es básica en este proyecto, pues se trata de desarrollar un programa para ella, encuadrada dentro del proyecto de la Comisión Europea. Además, es quien nos proporciona el ancho de banda disponible, por lo que es indispensable para que el proyecto funcione.

Por último, utilizamos iptables [9][10] para desviar nuestros paquetes hacia nuestro escuchador. Decidimos usarlo debido a su simplicidad como firewall y por estar disponible en todas las versiones de Linux con Kernel a partir de la versión 2.3.

A continuación, procederemos a explicar las diferentes tecnologías elegidas, de manera que quede más claro cómo funcionan y por qué se eligieron.

2.3.1 SIP (Session Initiation Protocol)

SIP se trata de un protocolo de inicio de sesión, desarrollado inicialmente por el grupo MMUSIC (Multi-Party Multimedia Session Control) en 1997, alcanzando nivel de estándar en 1999, en la RFC 2543 [11]. Actualmente se define en la RFC 3261, mantenido y actualizado por el grupo SIPCORE en el IETF. Lo primero que debe quedar claro sobre SIP es que se trata de un protocolo para establecer sesiones multimedia, pero no realiza esta comunicación; para eso se complementa con otros protocolos como SDP (descripción de la sesión), RTP (transmisión de la información), etc., que completan el servicio. Tampoco reserva recursos para la comunicación, para lo que se pueden usar otros protocolos como RSVP o extensiones a SIP que sí lo permiten. SIP crea, modifica y finaliza sesiones sobre cualquier transporte e independientemente del tipo de sesión que se haya establecido. Además, puede añadir clientes a una sesión ya establecida, o modificar las condiciones en cualquier momento. SIP realiza funciones básicas para establecer y finalizar las comunicaciones multimedia:

- Localización del usuario: determinación del sistema final (teléfono, PC…) que va a ser utilizado.
- Disponibilidad del usuario: comprueba si el usuario llamado está disponible para establecer la comunicación.
- Capacidades del usuario: elección de las características multimedia.
- Ajustes de la sesión: establecimiento de los parámetros a usar tanto en el llamante como en el llamado.

- Manejo de la sesión: establecer, modificar y terminar sesiones y llamar a otros servicios.

Existen varios elementos dentro de las comunicaciones SIP:

- Agente de Usuario (UA): en nuestro caso, será lo único necesario, y lo que utilizaremos. Se trata de un dispositivo final que implementa SIP (teléfonos, PC, Smart TV…). Se utiliza para establecer y finalizar sesiones con otros usuarios, y mantiene registro de las diferentes sesiones que tiene iniciadas. Contiene una aplicación cliente (UAC), que genera solicitudes, y otra servidor (UAS), que genera respuestas. Durante la sesión suelen trabajar de ambas formas.
- Servidores: existen diferentes tipos de servidores en SIP, siendo los más comunes los Proxy; también existen otros como los Redirect Server y los Register Server. Dado que no son necesarios y no se utilizarán en el proyecto, no entraremos en detalle en sus características.
- Back-to-back User Agent (B2BUA): se trata de un tipo especial de UA, que recibe solicitudes y las procesa como un UAS, y envía solicitudes hacia otros UAS como un UAC.

Las solicitudes SIP tienen una estructura determinada, que siempre será:

<Method> <Request-URI> <SIP-Version>

Donde:

- **Method** especifica el tipo de solicitud (ej. INVITE).
- **Request-URI** indica el destinatario de la solicitud (ej. Una URI SIP).
- **SIP-Version** indica la versión de SIP (generalmente, "SIP/2.0").

Existen dos tipos de solicitudes, las de SIP originales y las de sus extensiones. Nos centraremos en las originales, aunque comentaremos cuáles son las de las extensiones. Las solicitudes originales son:

- **INIVTE:** utilizada para establecer sesiones multimedia. Existe una solicitud **re-INVITE** utilizada para cambiar la descripción de la sesión, enviando una nueva solicitud "INVITE" una vez la sesión está establecida. Pueden modificar la sesión tanto el llamante como el llamado, y solo se aplica el cambio al recibirse una solicitud del tipo 2xx.
- **REGISTER:** informa a un Register Server de la localización de un usuario.
- **ACK:** respuesta definitiva a una solicitud de tipo INVITE.
- **CANCEL:** cancela solicitudes INVITE pendientes (aún no se ha enviado un ACK).
- **OPTIONS:** sirve para solicitar las capacidades multimedia de un UA.

En cuanto a las solicitudes de las diferentes extensiones, existen: **SUBSCRIBE, NOTIFY, PUBLISH, REFER, MESSAGE, INFO, PRACK** y **UPDATE.** (son las que vienen en clase, faltara alguna más??MIRARLO)

Un ejemplo de la estructura de una solicitud, en este caso un INVITE, se muestra a continuación:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

Ilustración 1- Ejemplo de estructura de solicitud INVITE [1]

Como se puede ver, la estructura de la solicitud es la indicada anteriormente, siendo en este caso una solicitud de tipo INIVITE con destino bob@biloxi.com y con versión de SIP 2.0. Después vienen una serie de campos de cabecera que a continuación explicamos (los necesarios o más comunes, existen más):

- **Via**: se trata de un registro de los proxies atravesados por la solicitud, de manera que la respuesta atraviesa los mismo proxies pero en sentido inverso. Al final, contiene la dirección en la que el cliente espera recibir las respuestas futuras. Pueden existir varias cabeceras "Via". El parámetro **branch** identifica la transacción.
- **Max-Forwards**: determina el número de saltos que puede dar la solicitud antes de llegar al servidor, de manera que cada vez que pasa por un proxy disminuye su valor en uno. En el caso de llegar a cero se descarta la solicitud.
- **To**: contiene el nombre y la URI del destinatario final de la solicitud.
- **From**: indica quién inició la solicitud. Debe incluir su URI, aunque también puede incluir el nombre. El parámetro **tag** es una cadena aleatoria de números añadidos por el terminal para tareas de identificación.
- **Call-ID**: se trata de un identificador único de una solicitud INVITE y de todos los registros realizados por ese cliente. Está formado por una secuencia aleatoria y el nombre del host o IP del terminal.
- **CSeq**: incluye un número aleatorio (en un primer caso) y el tipo de solicitud. Cada nueva solicitud incrementa su valor en uno a partir del original. Tiene tres propósitos principales:
  - Ordenar solicitudes.
  - Diferenciar nuevas solicitudes de retransmisiones.
  - Asociar cada respuesta a su solicitud.
- **Contact**: contiene una URI directa hacia el origen de la solicitud. Mientras que la cabecera "Via" indicaba dónde se debían enviar las respuestas, la cabecera "Contact" determina dónde se deben enviar futuras solicitudes.
- **Content-Type**: contiene una descripción del cuerpo del mensaje.
- **Content-Length**: longitud en bytes del cuerpo del mensaje.
- **Expires**: determina el intervalo de tiempo en el que la solicitud o el contenido del mensaje son válidos. Puede tratarse de un tiempo en segundos o de una fecha concreta.

Las solicitudes generadas por un UA deben obtener respuesta. Existen varios tipos, como se indica en la siguiente tabla:

CÓDIGO	TIPO DE RESPUESTA	DESCRIPCIÓN
100-199	Información	Mensaje provisional que indica el estado de la sesión mientras se procesa la solicitud
200-299	Éxito	Mensaje de éxito en el inicio de la sesión
300-399	Redirección	El cliente debería redirigir la solicitud a alguno de los servidores devueltos por el servidor inicial
400-499	Error de cliente	El cliente debe modificar su solicitud antes de volver a enviarla
500-599	Error de servidor	Existe un problema interno en el servidor, se puede reenviar directamente la solicitud a otro
600-699	Error global	El servidor tiene información sobre el cliente que le hace rechazarlo, no a esa solicitud en particular. No debería reintentar la solicitud ni con ese ni otros servidores

Tabla 1 – Tabla de códigos de respuesta y su descripción

Un ejemplo de una llamada que atraviesa dos servidores sería el siguiente:

Ilustración 2 – Ejemplo de llamada que atraviesa dos servidores [1]

En este caso, Alice querría establecer una comunicación con Bob, por lo que envía una solicitud de tipo INVITE. Los proxies le devuelven una respuesta de tipo "Trying" mientras esperan la respuesta de Bob. Una vez Bob ha recibido la llamada, se envía una respuesta "Ringing", que significa que ha recibido la solicitud. Al descolgar Bob, se envía un "OK", lo que significa que ha aceptado. En el caso de colgar, se recibiría una solicitud de tipo "CANCEL", y no se establecería la comunicación. Alice envía un "ACK" que establece definitivamente la sesión, y se inicia la comunicación multimedia. Como se ha comentado, esto ya no tiene que ver con SIP, que simplemente sirve para establecer, modificar y terminar la sesión. Generalmente se tratará de una comunicación usando el protocolo RTP/RTCP.

Durante la comunicación multimedia, tanto Alice como Bob podrían enviar una nueva solicitud de INVITE en el que cambien las características de la sesión. Finalmente, Bob envía una solicitud "BYE" para terminar la sesión, que es respondida afirmativamente con un "OK" por Alice. Debe quedar claro que aunque Alice haya iniciado la sesión, no es necesario que la finalice Bob; simplemente se trata de un ejemplo en el que se trata de ilustrar la estructura de una sesión SIP.

2.3.2 SDP (Session Description Protocol)

SDP fue definido por el grupo de trabajo del IETF MMUSIC en la RFC 2327, que fue actualizado por la RFC 3266 y en la actualidad la RFC 4566. Se trata de un protocolo para describir y elegir las características multimedia que se van a utilizar en una sesión. Simplemente describe la sesión, sin tener en cuenta de qué tipo es esta, pues la información proporcionada por SDP se puede combinar con diferentes protocolos: Session Announcement Protocol, Session Initiation Protocol, Real Time Streaming Protocol, e-mail utilizando las extensiones MIME e Hypertext Transport Protocol.

En el caso que a nosotros nos interesa, es decir, en combinación con SIP, se utiliza un modelo de oferta/respuesta, en el que un usuario genera una carga SDP (en el INVITE) y se la envía a otro usuario o servidor, que enviará una respuesta SDP (en el OK) al primer usuario en el que se determina qué características se van a usar. El modelo oferta/respuesta de SDP viene especificado en la RFC 3264 [12].

La carga SDP debe contener la siguiente información:

- Nombre y propósito de la sesión.
- Tiempo durante el cual la sesión está activa.
- La información multimedia de la sesión.
- Información necesaria para recibir la información multimedia (direcciones, puertos, etc.).

También puede contener otra información como el ancho de banda que debe usar la sesión o información de contacto de la persona responsable de la sesión.

La información multimedia que debe contener, en nuestro caso, será:

- El tipo de contenido multimedia (audio, vídeo, etc.).
- El protocolo de transporte.
- El formato del contenido multimedia.
- La dirección y el puerto de destino.

Además, se debe incluir información sobre el tiempo, tanto cuando empieza y acaba la sesión como si existe algún criterio de repetición (todos los días a una hora, cada mes, etc.).

Una descripción SDP está formado por una serie de líneas del tipo:

<type>=<value>

Donde <type> es un carácter único y <value> una cadena de texto que dependerá de cada valor de <type>. Nunca se debe incluir un espacio a ninguno de los lados del "=". La estructura está formada por una sección en la que se describe la información de la sesión ("v=") seguida, si procede, de las secciones de información multimedia necesarias ("m="). Puede no existir ninguna sección multimedia. Algunos de los campos a incluir son obligatorios y otros opcionales. Estos últimos los marcaremos con un "*":

- Descripción de sesión:

TIPO	DESCRIPCIÓN
v	Versión del protocolo
o	Creador e identificador de la sesión
s	Nombre de la sesión
i*	Información de la sesión
u*	URI donde se describe la sesión
e*	Dirección de email
p*	Número de teléfono
c*	Información de conexión
b*	Información de ancho de banda
t	Tiempo durante el que la sesión está activa
r*	Cuándo hay que repetir la sesión
z*	Ajustes de zona horaria
k*	Código de encriptación
a*	Atributos de la sesión

Tabla 2 – Tipos de descriptores de sesión y su significado [3]

- Descripción de la información multimedia, si la hubiese:

TIPO	DESCRIPCIÓN
m	Tipo de información multimedia y de transporte
i*	Título
c*	Información de conexión (opcional si está incluido en la

	descripción de la sesión
b*	Información de ancho de banda
k*	Código de encriptación
a*	Atributos multimedia

Tabla 3 – Descriptores de la información multimedia [3]

Cualquier otra letra que se utilice será directamente ignorada. Los tipos "c" y "a" que se especifiquen en el nivel de sesión serán válidos para todas las características multimedia de la sesión excepto sí se sobrescriben en el nivel multimedia.

El esquema básico típico de la descripción SDP, de cara al trabajo que se va a realizar, es:

v=0 (única versión de momento)

o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>

s= <session name>

c=<nettype> <addrtype> <connection-address>

t=<start-time> <stop-time>

Descripción de sesión

m=<media> <port> <proto> <fmt>…

a=rtpmap:<payload type> <encoding name>/<clock rate> [/<encoding parameters>]

m=<media> <port> <proto> <fmt>…

a= rtpmap:<payload type> <encoding name>/<clock rate> [/<encoding parameters>]

Descripción multimedia

Debe quedar claro que este esquema no es fijo, como se vio antes hay muchas más opciones, y otras de las que se han elegido no son obligatorias. En cambio, este esquema es el más común en el caso de una comunicación multimedia con audio y vídeo de dos participantes, que es el supuesto con el que trabajaremos en este proyecto. Por tanto, vamos a explicar qué significan los diferentes valores de los que consta cada campo:

- **v**: un único valor. Ahora mismo, solo existe la versión 0.
- **o**: viene determinado por seis parámetros:
  - **username**: nombre del host de origen, o "-" en el caso de que no exista.
  - **sess-id**: es una cadena numérica de manera que, en conjunto con los demás parámetros, se forme un identificador único. Aunque no hay una forma obligatoria de definirlo, se recomienda utilizar una marca de tiempo NTP (Network Time Protocol).
  - **sess-version**: número que describe la sesión. También se recomienda utilizar la marca de tiempo NTP, por lo que suele coincidir con <sess-id> en un principio, pero en este caso se actualiza cuando se modifica la sesión.

    - **nettype**: define el tipo de red sobre el que se trabaja. Actualmente se utiliza "IN" para referirse a "Internet", pero se podrían registrar otros en el futuro.
    - **addrtype**: indica el tipo de dirección que va a continuación. Actualmente, puede ser "IP4" o "IP6", aunque podrían añadirse nuevos tipos en el futuro.
    - **Unicast-address**: dirección de la máquina que creó la sesión. Puede tratarse del nombre del dominio en el que se encuentra o la dirección IP (ya sea IPv4 o IPv6) de la máquina.
- **s**: indica, textualmente, el nombre de la sesión. Solo puede haber uno por sesión y no puede estar vacío, aunque sí contener únicamente un espacio en blanco.
- **c**: una sesión debe contener al menos una "c" en cada descripción multimedia o una en la descripción de sesión. En el caso de existir una de sesión y otra multimedia, esta segunda sobrescribe a la primera. En cuanto a sus parámetros:
    - **nettype**: como en el caso anterior, se trata del tipo de red. Actualmente, únicamente está definido "IN", que se refiere a Internet.
    - **addrtype**: define el tipo de dirección que se va a utilizar. Se define "IP4" e "IP6" actualmente.
    - **connection-address**: representa la dirección, unicast o multicast, IPv4 o IPv6 que se va a utilizar para la comunicación. En función del tipo de dirección, pueden existir parámetros adicionales, como por ejemplo el TTL en el caso de conexiones IPv4 multicast.
- **t**: especifica el tiempo durante el cual está activa la sesión, mediante un momento de inicio ("**start-time**") y uno de fin (**stop-time**"). Puede existir más de una línea correspondiente al tiempo, si se inicia en diferentes momentos y no hay un patrón de repetición. Los campos de tiempo representan, en segundos, el tiempo pasado desde 1900, conocido como Network Time Protocol. Tanto el momento de inicio como el de fin pueden tener valor cero. En el caso de valer cero "stop-time", la sesión estará activa desde el momento en que se llegue al "start-time". Si este también es cero, la sesión será permanente. No se recomienda usar este tipo de sesiones.
- **m**: pueden existir varias descripciones multimedia de la sesión. Los diferentes subcampos significan:
    - **media**: indica el tipo de contenido multimedia que nos vamos a encontrar. Ahora mismo, puede tomar valores "audio", "video", "text", "application" y "message", aunque se podrían añadir nuevos tipos.
    - **port**: puerto en el que se va a recibir el contenido multimedia. En el caso de que otros protocolos multimedia como RTCP (Real Time Control Protocol) necesiten otro puerto, este se calculará algorítmicamente o se especificará en otro atributo, del tipo "a=". En el caso de ser necesario más de un puerto, se utilizará la notación "<port>/<number of ports>. Cómo se repartan esos puertos dependerá de cada protocolo; por ejemplo, en el caso de RTP, solo se utilizan los puertos pares para RTP y los impares para RTCP, por lo que en el caso de necesitar dos puertos, se elegirán el indicado como primero y el siguiente par como segundo, dejando los impares para RTCP. En el caso de que se hubiesen especificado varias direcciones en el campo

"c=", y varios puertos en "m=", se asociarán el primero con el primero, el segundo con el segundo, etc.
    - **proto**: especifica el protocolo de transporte. Están definidos los siguientes:
        - **udp**: no se especifica el protocolo que trabaja sobre UDP [20].
        - **RTP/AVP**: RTP Profile for Audio and Video Conferences with Minimal Control sobre UDP.
        - **RTP/SAVP**: Secure Real-time Transport Protocol sobre UDP.
    - **fmt**: se trata de una descripción del formato multimedia. Todos los campos a partir del primer "fmt" se referirán a esta descripción, y su interpretación dependerá del campo "proto":
        - En el caso de tratarse de "RTP/AVP" o "RTP/SAVP", se tratará del número que identifica el tipo de datos RTP. En el caso de existir varios, todos los formatos indicados podrían usarse en la sesión, siendo el primero el que se usaría por defecto. Mediante el uso de "a=rtpmap:" se asocian estos números al nombre de la codificación multimedia correspondiete.
        - Si se trata de "udp", en este campo se debe describir el formato multimedia dentro de los tipos "audio", "video", "text", "application" o "message".
- **a**: define los atributos multimedia. Existen numerosas opciones diferentes, pero para nuestro trabajo solo nos interesa la de tipo "**a=rtpmap:**", por lo que será la única que describamos:
    - **payload type**: se trata del mismo número correspondiente al atributo "fmt" en la descripción multimedia.
    - **encoding name**: se trata del nombre del formato multimedia asociado a cada "payload type".
    - **clock rate**: frecuencia a la que trabaja el formato especificado.
    - **encoding parameters**: en el caso de streams de audio especifica el número de canales (si es uno, no es necesario). En el caso de vídeo, no existe ninguna opción para añadir en este campo.

De esta manera, mediante una descripción de sesión y una multimedia para cada tipo (audio y vídeo), con diferentes formatos dentro de ellas, tendríamos el escenario típico para, por ejemplo, una videollamada. Se trata de uno de los escenarios más clásicos y también sencillos, aunque muestran a la perfección las capacidades y la sencillez de SDP como protocolo para definir sesiones multimedia.

2.3.3 RTP/RTCP (Real Time Protocol/RTP Control Protocol)

La documentación sobre RTP está recogida en las RFC 3550 y 3551, que dejan obsoleta la RFC 1889 [13]. RFC proporciona un servicio extremo a extremo para servicios en tiempo real, como audio y vídeo interactivos. RTP no proporciona ningún mecanismo para asegurar la entrega a tiempo ni calidad de servicio (QoS). Los paquetes RTP incluyen números de secuencia que permiten conocer el orden en el que fueron enviados, o saber la posición de un paquete sin necesidad de decodificar los demás. RTP debe estudiarse junto con

RTCP; RTP se encarga de llevar la información en tiempo real, mientras que RTCP se encarga de proporcionar información sobre la sesión y cierta calidad de servicio.

Existen dos tipos de documentos que complementan la información sobre RTP:

- **Perfiles**, que definen una serie de tipos de datos y los asignan a sus códigos correspondientes, además de poder añadir extensiones o modificaciones específicas para cierto tipo de aplicaciones. Para este trabajo, necesitaremos profundizar en el perfil para audio y vídeo definido en la RFC 3551.
- **Especificación de tipos de datos**, definen cómo un tipo de datos en particular debe ser transportado por RTP.

Es importante tener en cuenta que, en el caso de que se esté transmitiendo audio y vídeo, cada uno de ellos será una sesión RTP/RTCP diferente, que utilizarán distintos puertos. Esto permite que, si alguno de los participante no puede/desea recibir ambos flujos de datos, pueda participar en la sesión con los datos deseados.

Es necesario reservar, al menos, dos puertos UDP para recibir paquetes: el puerto par servirá para recibir los paquetes RTP, mientras que el impar inmediatamente, superior se utilizará para los paquetes RTCP.

Existen dos tipos de dispositivos intermedios en una sesión RTP que hacen que ésta sea más eficiente:

- **Mezcladores**: reciben datos de una o más fuentes, y tras combinarlos y/o cambiarlos de formato, los reenvían. Utiliza las *timestamps* para sincronizar y temporizar el flujo de salida en el que se combinan los diferentes tipos de entradas. Además, si hubiese algunos clientes que tuviesen un menor ancho de banda o no soportasen algún tipo de comunicación, con el mezclador se puede cambiar el formato del flujo de datos, de manera que no todos los participantes tienen que adoptar las mismas características. El mezclador envía el flujo de datos a una o más fuentes de manera unicast o multicast.
- **Traductores**: sirven para cambiar el formato de un flujo de datos, como en el caso anterior, y también se puede utilizar para atravesar firewalls que no permiten el tráfico multicast a través de un túnel unicast.

La cabecera de un paquete RTP es la siguiente:

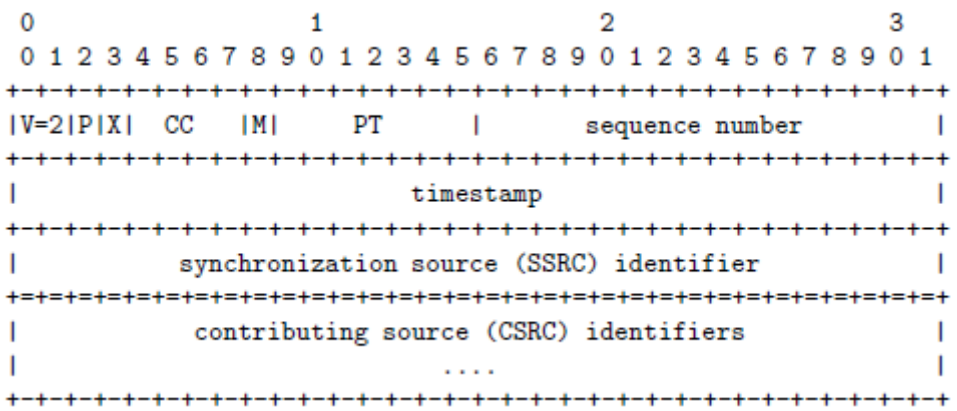


Ilustración 3 – Cabecera de paquete RTP [7]

Donde:

- **Version, V (2 bits)**: identifica la versión de RTP. En la actualidad siempre se utiliza 2.
- **Padding, P (1 bit)**: si vale 1, el paquete contiene relleno, debido a que algunos algoritmos de encriptación trabajan con un tamaño de bloque fijo, por ejemplo. En este caso, el último byte del relleno indicará cuántos bytes deben ser ignorados.
- **Extension, X (1 bit)**: si vale 1, la cabecera debe ir seguida por una extensión.
- **CSRC count, CC (4 bits)**: la cuenta de contribuyentes indica el número de identificadores CSRC que siguen a la cabecera.
- **Marker, M (1 bit)**: su interpretación depende de los diferentes perfiles.
- **Payload type, PT (7 bits)**: identifica el formato de la carga RTP y su interpretación por la aplicación. El perfil debe tener una asociación entre los códigos y los formatos correspondientes.
- **Sequence number (16 bits)**: el primer valor se calcula aleatoriamente. Después, su valor se incrementa en uno con cada paquete RTP enviado, de manera que el receptor pueda colocar los paquetes o detectar los perdidos.
- **Timestamp (32 bits)**: marca el instante de muestreo del primer octeto de datos del paquete RTP. Para su generación se utiliza un reloj local que se incrementa de manera monótona y lineal en el tiempo, siendo el valor inicial aleatorio. La frecuencia del reloj depende del formato de la carga RTP, definida en el perfil correspondiente.
- **Synchronization Source Identifier, SSRC (32 bits)**: se trata de un identificador único para una fuente RTP dentro de una sesión, de manera que un receptor agrupa todos los paquetes de una sesión RTP que recibe con el mismo SSRC para poder reproducirlo. Se genera aleatoriamente por la fuente para cada sesión.
- **Contributing Source Identifier, CSRC (0 a 15 ítems, 32 bit cada uno)**: cuando el paquete pasa por un mezclador, añade el SSRC de cada fuente que aporta al contenido del paquete, hasta un máximo de 15.

El funcionamiento de RTCP se basa en la transmisión periódica de paquetes de control a los diferentes participantes de la sesión con el mismo mecanismo de distribución que los paquetes de datos RTP. Sus funciones son:

- Proporcionar información sobre la calidad de la distribución de paquetes, permitiendo tomar medidas relacionadas con control de congestión.
- Transportar información de identificación de la fuente, “Canonical Name” (CNAME), de manera que el resto de participantes puedan identificarla y asociar diferentes streams del mismo participante. Su función es similar al SSRC de RTP, pero éste puede cambiar en el caso de existir un conflicto o se reinicia el programa, CNAME es propio de cada participante.
- Dado que todos los participantes envían paquetes RTCP periódicamente, debe existir una función de control de la tasa de envío de éstos, de manera que no se congestione la red con ellos.
- Opcionalmente, se puede incluir información de control de la sesión, como número de participantes, identificación de ellos, etc.

Existen diferentes tipos de paquetes RTCP en función de la información que contengan:

- **SR:** Sender report, estadísticas de transmisión y recepción de transmisores activos.
- **RR:** Receiver report, estadísticas de recepción para participantes que no son emisores activos y en combinación con SR en transmisores activos en más de 31 fuentes.
- **SDES:** Descripción de la fuente, incluyendo CNAME
- **BYE:** final de la participación, indicando, si se desea, el motivo por el que se abandona la sesión.
- **APP:** funciones específicas de la aplicación.

Estos paquetes se combinan en un paquete compuesto para su envío. En éstos siempre debe haber un SR o un RR, pudiendo haber más, y también un "SDES" que contenga al menos un CNAME. Además, pueden llevar cualquier otro paquete RTCP sin importar el orden, salvo el último que será "BYE" en el caso de haberlo. Por tanto, un paquete RTCP enviado deberá tener la siguiente estructura:

1. **Prefijo de encriptación**, en el caso de haberlo.
2. **SR o RR**, incluso aunque sea el primer paquete que se envía o si el otro paquete con el que se envía es un "BYE".
3. **Otros RR**, en el caso de existir por estar recibiendo más de 31 fuentes.
4. **SDES**, con al menos un CNAME, aunque puede llevar más información.
5. **BYE o APP**, u otros futuros tipos de paquete, se incluirán aquí sin importar el orden salvo en el caso del "BYE" que será el último.

De esta manera, podríamos obtener un paquete como el del siguiente ejemplo:

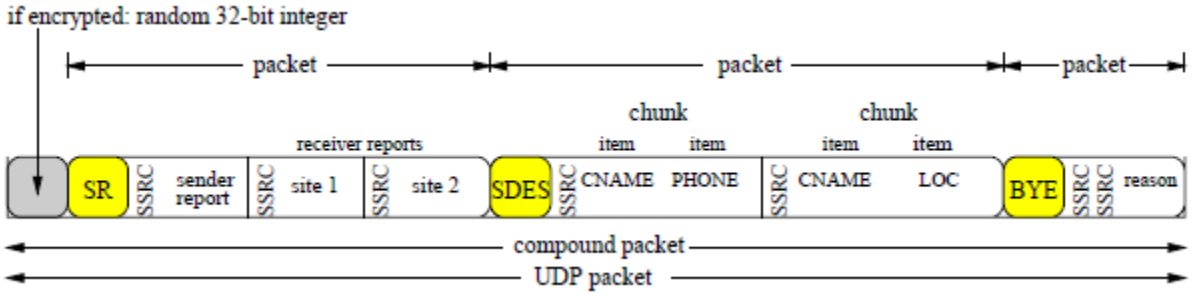


Ilustración 4 – Ejemplo de paquete RTCP [7]

Como se puede ver, en el paquete combinado están incluidos tres paquetes, de los cuales el SR y el SDES son obligatorios.

Como ya se comentó anteriormente, es muy importante no enviar demasiados paquetes RTCP para no saturar la red con mensajes de control, y también que la tasa utilizada por estos mensajes sea conocida de manera que se pueda incluir en el ancho de banda reservado para la sesión, si lo hubiese. Por tanto, se recomienda que no más del 5% del tráfico sea RTCP, y que el 25% de ese tráfico este reservado para los usuarios que envían información (siempre y cuando el número de usuarios que la envían sea inferior a ese 25%), de manera que nuevos

usuarios puedan recibir cuanto antes el CNAME de estos emisores. Se recomienda que el mínimo intervalo de envío de paquetes RTCP sea de 5 segundos.

Los paquetes SR y RR nos proporcionan información sobre la calidad de la sesión. Ambos paquetes son muy parecidos, la única diferencia está en que los SR incluyen además información sobre el emisor. En primer lugar, trataremos la estructura de un paquete SR:

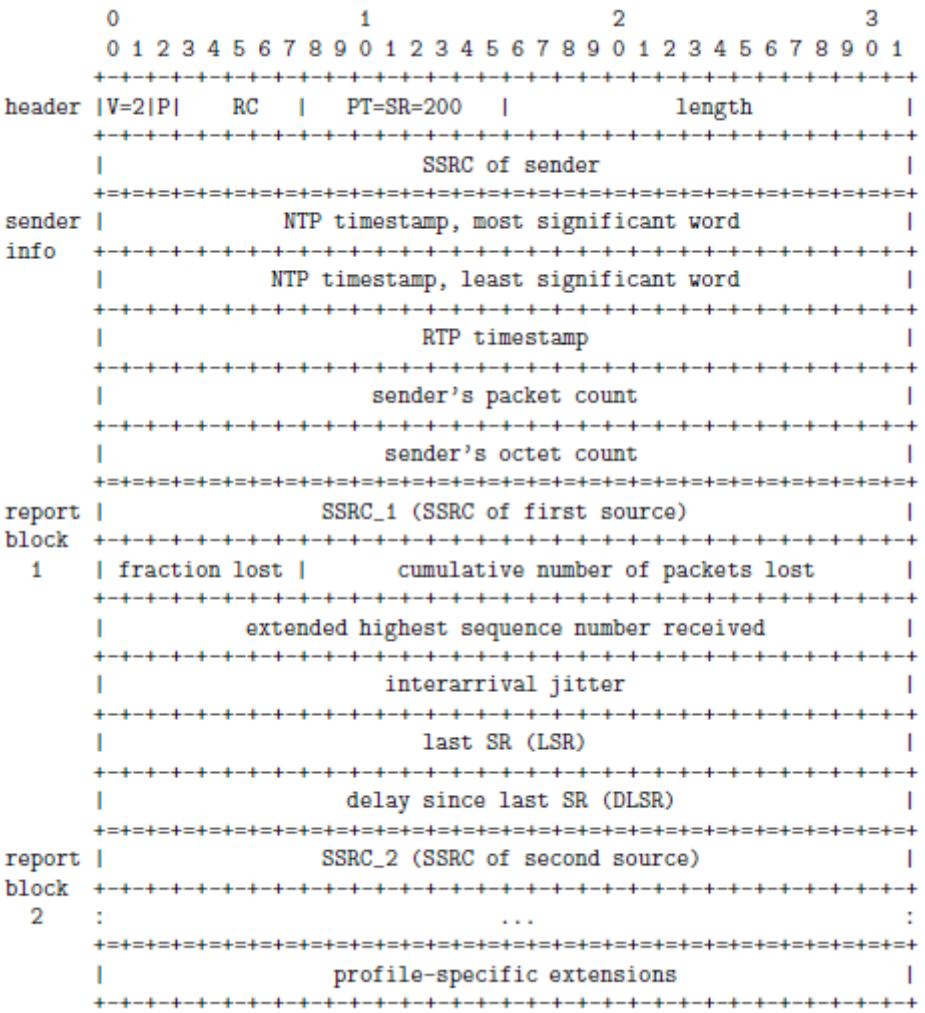


Ilustración 5 – Estructura paquete SR [7]

Como se puede observar existen tres zonas diferenciadas: cabecera, información de emisor y bloque de reportes. Además, existe una cuarta parte opcional correspondiente a las extensiones de perfil. Empezaremos por las cabeceras, que tienen una longitud de ocho octetos:

- **Version, V (2 bits)**: versión de RTP, la misma para los paquetes RTCP. Actualmente versión 2.
- **Padding, P (1 bit)**: en el caso de estar a 1 existirán algunos octetos de relleno al final. El último indica cuántos deben ser ignorados.
- **Reception report count, RC (5 bits)**: número de RR que contiene el paquete. Puede tener valor cero.

- **Packet type, PT (8 bits)**: en los paquetes SR, constante de valor 200 que lo identifica como paquete RTCP SR.
- **Length (16 bits)**: longitud del paquete, incluyendo las cabeceras y el relleno si lo hubiese.
- **SSRC (32 bits)**: SSRC del emisor del paquete SR.

En segundo lugar, explicaremos los diferentes campos de la información del emisor, que tienen un total de veinte octetos:

- **NTP timestamp (64 bits)**: indica el momento en el que se envía el paquete (fecha y hora) de manera absoluta, representado en segundos el tiempo transcurrido desde el 1 de enero de 1900.
- **RTP timestamp (32 bits)**: representa las mismas unidades que el "NTP timestamp" pero en las unidades que se utilizan en los paquetes RTP, según el reloj local.
- **Sender's paquet count (32 bits)**: número total de paquetes enviados por el emisor desde que comenzó a transmitir hasta que envía el reporte. Se reiniciará en el caso de que cambie su identificador SSRC.
- **Sender's octet count (32 bits)**: el número total de octetos de carga enviados (sin contar cabeceras ni relleno) transmitidos en los paquetes RTP desde que el emisor inició la transmisión hasta la actualidad. Debe reiniciarse si se cambia el identificador SSRC. Estos dos campos en común pueden utilizarse para calcular la tasa de datos media por paquete.

La tercera y última parte obligatoria contiene la información sobre la recepción de paquetes, tantas como fuentes esté escuchando el usuario. Cada bloque de reporte nos da estadísticas sobre la recepción de paquetes RTP de cada fuente, por lo que tendremos un bloque para cada una. El significado de los diferentes campos y estadísticas es:

- **SSRC_n, source identifier (32 bits)**: identificador de la fuente al que pertenece la información. "n" se refiere a cada uno de los bloques recibidos.
- **Fraction lost (8 bits)**: fracción de paquetes de la fuente que se han perdido desde el anterior paquete SR o RR. Divide el número de paquetes recibidos entre los esperados.
- **Cumulative number of packets lost (24 bits)**: número total de paquetes RTP enviados por la fuente que se han perdido. Se calcula a partir del número de paquetes esperados y los que realmente se reciben, incluyendo los duplicados o tardíos.
- **Extended highest sequence number received (32 bits)**: se divide en dos partes: los 16 bits más pequeños representan el número de secuencia más alto recibido en un paquete RTP enviado por la fuente, mientras que los 16 más grandes lo extienden con el número de ciclos de número de secuencia ya pasados.
- **Interarrival jitter (32 bits)**: se trata de una estimación de la variación del tiempo entre paquetes RTP, medido en función de las marcas de tiempo RTP. Se trata de la desviación media entre el momento de envío y de llegada de un par de paquetes, según la siguiente ecuación, donde *S* es "RTP timestamp" y *R* es el momento de llegada en las mismas unidades:

$$D(i,j) = (Rj - Sj) - (Ri - Si)$$

Este valor se debería calcular de manera continua cada vez que se reciba un paquete, en función del valor de *D* de ese paquete y del anterior, según la fórmula:

$$J(i) = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16}$$

Cada vez que se recibe un RR o un SR el valor del jitter se actualiza. El parámetro 1/16 se utiliza para reducir el ruido y mantener un ratio de convergencia razonable. Siempre se debe utilizar esta fórmula para calcular el jitter.

- **Last SR timestamp, LSR (32 bits)**: el valor de los 32 primeros bits del NTP timestamp del anterior paquete SR. En el caso de no existir ninguno, su valor será cero.
- **Delay since last SR, DLSR (32 bits)**: retardo desde que se recibió el último SR hasta que se envía el bloque de reporte, siendo sus unidades 1/65536 segundos. En el caso de no haberse recibido ningún paquete SR, valdrá cero.
  Podemos calcular el round-trip a partir de *A*, momento en el que se recibe el bloque de reporte, LSR y DLSR. El tiempo total podemos obtenerlo restando *A-LSR*, y después el retardo de round-trip restando el DSLR, siendo finalmente *A-LSR-DSLR*. Un ejemplo del cálculo de estos datos se puede observar en la siguiente figura:

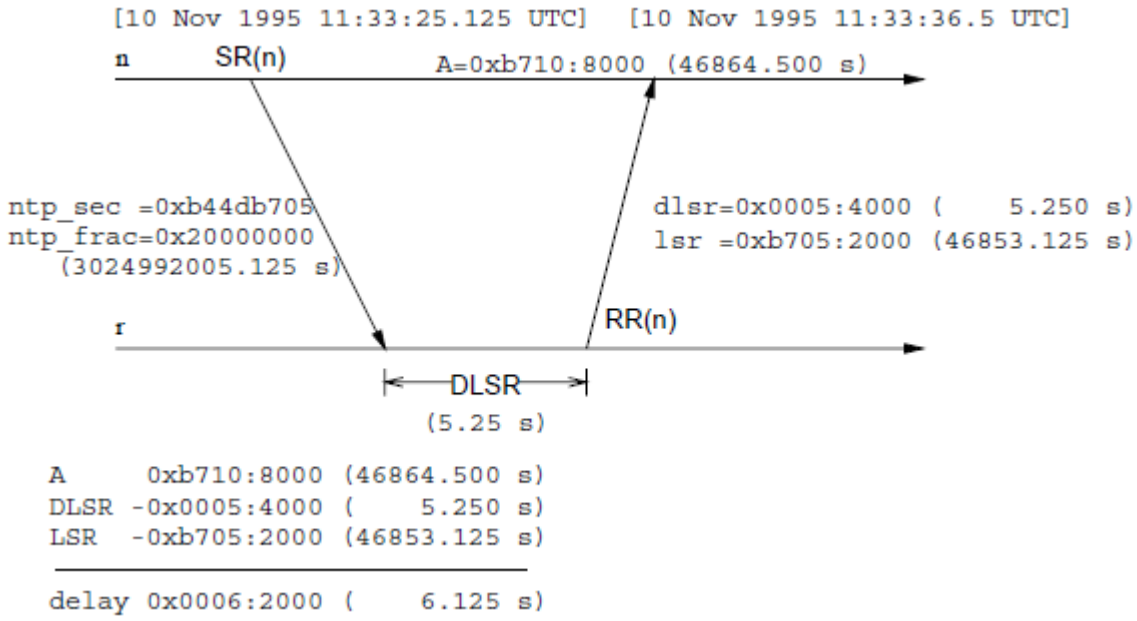


Ilustración 6 – Cálculo DSLR [7]

Después de los bloques de reporte, pueden incluirse extensiones propias de cada perfil, pero se trata de campos opcionales dependiendo del perfil usado.

En el caso de tratarse de un paquete RR en vez de un SR, su estructura sería igual, únicamente no existiría la parte correspondiente a la información del emisor, y "PT" tendría valor 201, por tratarse de un paquete de tipo RR.

Las especificaciones sobre codificación de audio y vídeo vienen en un perfil específico que se define en la RFC 3551. Se especifican diferentes tipos de codificaciones y de carga, con el número de tipo correspondiente a cada uno.

En primer lugar, la siguiente tabla muestra las diferentes codificaciones de audio y sus características:

Nombre del códec	Número	Sample/ frame	Bits/ sample	Clock rate (Hz)	Ms/ frame	Ms/ packet	Ancho de banda (Kb/s)
DVI4	5/6/16/17	Sample	4	Variable	-	20	32
G722	9	Sample	8	8000	-	20	64
G723	4	Frame	-	8000	30	20	6.3/5.3
G726-40	Dinámico	Sample	5	8000	-	20	40
G726-32	Dinámico	Sample	4	8000	-	20	32
G726-24	Dinámico	Sample	3	8000	-	20	24
G726-16	Dinámico	Sample	2	8000	-	20	16
G728	15	Frame	-	8000	2.5	20	16
G729	18	Frame	-	8000	10	20	8
G729D	Dinámico	Frame	-	8000	10	20	6.4
G729E	Dinámico	Frame	-	8000	10	20	11.8
GSM	3	Frame	-	8000	20	20	13
GSM-EFR	Dinámico	Frame	-	8000	20	20	13
L16	10/11	Sample	16	44100	-	20	128/1411.2
LPC	7	Frame	-	8000	20	20	2.4
MPA	14	Frame	-	90000	Variable	-	16/22.05/24/ 32/44.1/48
PCMA	8	Sample	8	8000	-	20	64
PCMU	0	Sample	8	8000	-	20	64
QCELP	12	Frame	-	8000	20	20	6.8/4.7
VDVI	Dinámico	Sample	Variable	Variable	-	20	32

Tabla 4 – Codecs audio estándar RTP [7]

En cuanto al vídeo, el ancho de banda requerido por cada uno de los codecs es más complejo, pues no se trata de valores fijos, sino que depende de las características del vídeo, de la calidad final que deseemos, las pérdidas permitidas y el movimiento de la cámara como principales factores que determinan el bitrate de salida. Por tanto, hemos tratado de buscar información sobre los valores de cada uno de ellos, principalmente el valor aproximado en el caso de una videoconferencia, o el máximo que alcanzaría. Dado que no se trata de ningún tipo de valor oficial, salvo en H261, su fiabilidad no es máxima como en el caso del audio. Las codificaciones permitidas son las de la siguiente tabla:

Nombre del códec	Número	Clock rate (Hz)	Ancho de banda (Kb/s)
CelB	25	90000	(1)
JPEG	26	90000	7200(2)
nv	28	90000	~6000(3)
H261	31	90000	px64, máx. 1920(4)
MPV	32	90000	1500(5)
MP2T	33	90000	(1)
H263	34	90000	4096(6)

Tabla 5 - Codeces de vídeo y sus características

(1)Para estos dos formatos no encontramos datos sobre el ancho de banda requerido para su funcionamiento en ningún caso.

(2) Valor obtenido mediante el calculador de ancho de banda [14], para la cámara [15], que tiene una resolución D1 a 30fps.

(3)Como se desprende de [16], es entre dos y tres veces superior al generado por H261.

(4) La tasa serán 64 kbit/s por cada hilo, siendo el máximo de hilos 30, lo que nos genera un total de 1920 kb/s, según recomendación de la ITU-T Rec. H261 [17]

(5) Valor obtenido de la ISO/IEC 11172-2:1993 [18]

(6) Según la ITU-T Rec. H261, para un nivel 50, para imágenes CIF y más pequeñas [19]

Además de estos números correspondientes a cada carga, hay una serie de ellos reservados por seguridad o futuros codecs que se añadan a la normativa.

2.3.4 Iptables

Iptables se trata de un firewall vinculado directamente al Kernel de Linux, siendo parte del propio sistema operativo.

La función de un firewall es filtrar el tráfico entre, al menos, dos redes. Puede tratarse tanto de un dispositivo físico como un software instalado en otro dispositivo. Al recibir un

paquete, el firewall decide si este pasa, se descarta o se modifica. El esquema básico de cómo funciona iptables lo tenemos a continuación:

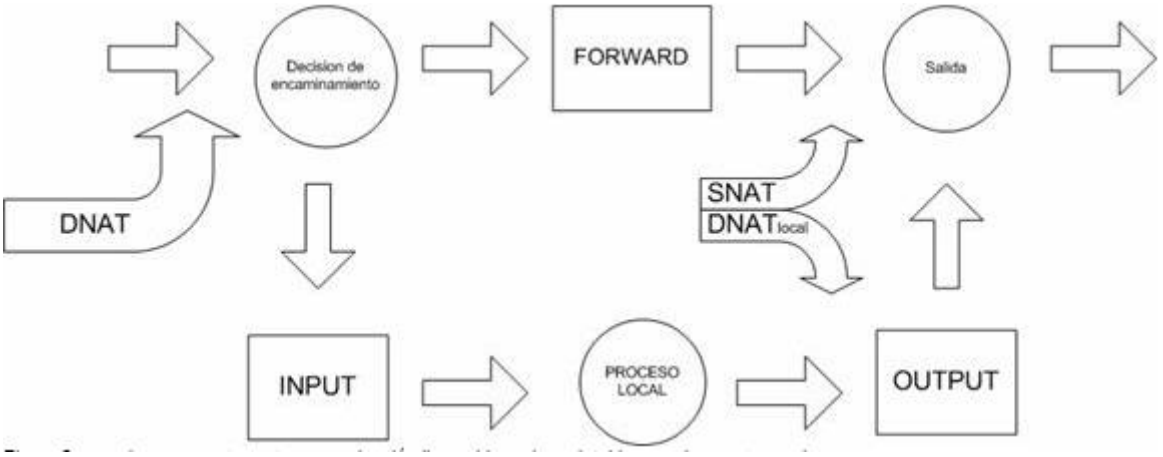


Ilustración 7 – Esquema funconamiento iptables [10]

Como se puede ver, tenemos tres tipos de reglas: "INPUT", "OUTPUT" y "FORWARD". INPUT y OUTPUT se utilizan para paquetes dirigidos a nuestra propia red, mientras que FORWARD se utiliza para paquetes dirigidos a otras redes o máquinas.

Además de estas reglas de filtrado, existen otras de tipo NAT para poder redirigir los paquetes a otra dirección o puerto, por ejemplo. Son las reglas PREROUTING y POSTROUTING, que permiten modificar el destino a la entrada y salida del paquete.

Por último, existen unas reglas de tipo MANGLE que permiten modificar el paquete, aunque estas son poco conocidas y utilizadas.

Al llegar un paquete, podemos filtrarlo de tres maneras:

- **ACCEPT:** acepta el tráfico indicado en la regla (de una dirección, a un puerto…)
- **DROP:** deniega el paso de paquetes con origen/destino definido en la regla.
- **QUEUE:** redirige el paquete a una cola destinada a ello en el espacio de usuario. Éste deberá tener un "queue handler" (manejador) que reciba los paquetes y un proceso a nivel de aplicación que decida qué se debe realizar con el paquete, después de realizar las acciones deseadas con él.

Con las políticas ACCEPT y DROP tenemos las funciones básicas de iptables, que se pueden aplicar en reglas individuales o en cadenas de ellas. Hay que tener en cuenta que en el caso de aplicar varias reglas, es muy importante el orden, pues se va leyendo secuencialmente y si a una misma regla se le aplican dos políticas diferentes, por ejemplo, la segunda sobrescribirá a la primera. Hemos explicado el caso especial QUEUE debido a que fue una opción que se planteó para el desarrollo de este trabajo, como se explicará más adelante.

Existen diferentes acciones que podemos realizar con las reglas, aunque las cuatro principales son las siguientes:

- Añadir una nueva regla, -A.
- Eliminar una reglas, -D.
- Eliminar todas las reglas de un tipo, -F.

- Listar todas las reglas aplicadas, -L.

Pese a que éstas son las principales, existen otras que permiten añadir o eliminar posiciones concretas en la cadena o mover las reglas dentro de ésta, por ejemplo, pero no entraremos en detalle en ellas por no ser necesarias para el desarrollo del proyecto ni básicas para la creación de un firewall iptables.

Para crear las reglas de filtrado los principales parámetros serán:

- **Dirección de origen y destino:** utilizaremos los comandos "-s", "--src" o "--source" para el origen y "-d", "--dst" o "--destination" para el destino, pudiendo especificar la dirección o direcciones deseadas tanto con su nombre como con su dirección ip.
- **Interfaz:** utilizaremos "-i" o "--in-interface" para las cadenas de tipo INPUT y FORWARD de entrada y "-o" o "--out-interface" para las de tipo OUTPUT y FORWARD de salida.
- **Protocolo:** mediante el uso de "-p" seguido de TCP [21] o UDP. Esto nos permite, posteriormente, especificar también el puerto de destino ("--destination-port" o "--dport") y el de origen ("--source-port" o "--sport"), además de algunas otras extensiones, como flags y opciones en el caso de TCP, aunque menos importantes.

En el caso del procesado PREROUTING y POSTROUTING, podemos realizar tres acciones básicas:

- **DNAT:** mediante este comando cambiaremos la dirección de destino.
- **SNAT:** lo utilizaremos para cambiar la dirección de origen.
- **REDIRECT:** su función será cambiar el puerto de destino. En este caso, tendremos que indicar primero cuál era el puerto de destino original con el comando "--dport" para posteriormente especificar el nuevo con "--to-port".

Con estos comandos y reglas estaremos preparados para crear un firewall básico con iptables que nos permita filtrar los paquetes que nos interesan.

2.3.5 Java

El lenguaje de programación Java [22][23] fue presentado en 1995 por Sun Mycrosystems, que fue posteriormente adquirida por Oracle. Se creó con el propósito de crear una plataforma sencilla, fiable, distribuida y en tiempo real, además de permitir que los desarrolladores escriban una vez su programa y puedan ejecutarlo en cualquier momento, pues no es necesario volver a recompilar el programa. Se trata de un lenguaje orientado a objetos y basado en clases. Se han consolidado como un lenguaje muy práctico para desarrollar aplicaciones de usuario final seguras en muchos entornos, sobre todo relacionados con entornos de red e Internet. Ha desarrollado herramientas especiales para diferentes plataformas, entre las que destacan entornos empresariales, sistemas embebidos y, aunque ha perdido fuerza, móviles, además de la versión estándar.

Gran parte del éxito de Java es su capacidad para ejecutarse en cualquier plataforma, pues se han creado máquinas de Java para los diferentes sistemas operativos del mercado.

Estas máquinas actúan como puente entre el sistema operativo y el programa, lo que hace posible que éste sea entendido sin ningún tipo de problema.

Otra de las fortalezas de Java, es el ecosistema formado por el gran número de desarrolladores que existen, lo que proporciona numerosas librerías especializadas en diferentes temas, además de una gran cantidad de documentación, comunidades y foros donde adquirir y compartir conocimientos, solucionar dudas y resolver problemas relacionados con cualquier aspecto del desarrollo del programa.

Este ecosistema fue el que nos impulsó a decidirnos por Java como lenguaje para este proyecto, pues existía un completo conjunto de librerías para el trabajo con paquetes SIP, lo que aumentaba las posibilidades de desarrollo.

JAIN-SIP [24] es una especificación de bajo nivel para la señalización de mensajes SIP, dando soporte para las diferentes RFC que componen SIP y los protocolos que se integran con él para completar su funcionamiento, como SDP y RTP. Contiene diversas librerías que permiten implementar prácticamente cualquier programa en cualquier entorno que trabaje con SIP, ya sean proxys, clientes, registers… Provee a los desarrolladores de un extenso Javadoc donde se registran todos los paquetes, clases, métodos y constructores que conforman el API completo de JSIP y proporciona un extenso soporte para el trabajo con los paquetes SIP.

2.3.6 Samknows Whitebox

SamKnows [2] es una empresa bastante nueva, que empezó su andadura en 2008 en Londres. Surgió gracias a un joven desarrollador, Sam Crawford, que decidió crear su propio router. Tras darse cuenta de que la mayoría de los usuarios no estaban satisfechos con su conexión a internet, decidió crear un router que la monitorizase, de la manera más fiable posible, en su punto de acceso, el router de su casa. De esta manera, pretendía eliminar posibles agentes externos que pudiesen afectar a la calidad de la conexión, como otras conexiones inalámbricas que interfieran con la propia, por ejemplo. Por tanto, desarrolló varios programas para su router de manera que pudiese monitorizar su conexión, y escribió un *paper* en el que explicaba su proyecto. Gracias a esto recibió muchas solicitudes, pero sobre todo sirvió para que Ofcom, el regulador de las telecomunicaciones en Reino Unido, se pusiese en contacto con él.

A partir de ahí, se iniciaron sus dos primeros proyectos para estudiar las redes de internet: Reino Unido y Estados Unidos, en 2009 y 2010 respectivamente, donde actualmente más de 10000 usuarios participan y colaboran en el proyecto.

En 2011, la Comisión Europea en Bruselas decidió contar con SamKnows para un ambicioso proyecto que tratase de analizar las comunicaciones en la Unión Europea, contando con una Whitebox que permitiese su uso en cualquier país de la Unión con, simplemente, conectarlo al router del usuario, e iniciase automáticamente el análisis, de manera totalmente transparente al usuario.

Tras el proyecto europeo también se han iniciado otros proyectos en Singapur y Brasil, extendiéndose de esta manera cada vez a más usuarios de zonas totalmente diferentes del mundo, con el fin de conseguir un mejor conocimiento sobre las redes de cada región, y que

esto en un futuro repercuta en una mejora de las redes y de la experiencia del usuario, y siguen ofreciéndose a nuevos proyectos en nuevos países.

SamKnows basa su proyecto en la transparencia, por lo que el código de las pruebas que realiza se trata de código abierto que se puede consultar, de manera que los resultados obtenidos por los participantes puedan ser comprobados si se desea con otras plataformas, y así demostrar su veracidad libremente. Tratan de que el sistema sea lo más abierto posible, publicando información sobre el proyecto y el código fuente de las pruebas realizadas.

Por otra parte, pretenden que cualquier persona que lo desee pueda ser voluntaria en los diferentes proyectos en desarrollo, sin necesidad de ser un usuario avanzado. Para ello, están trabajando en dos vías diferentes:

- Por un lado, están negociando con diferentes proveedores de hardware para introducir el software desarrollado directamente en los routers, de manera que no sea necesario un equipo extra, con la siguiente comodidad tanto por uso como por espacio.
- En los casos en los que aún no es posible, como en el proyecto europeo en el que se enmarca este trabajo, la Whitebox enviada se trata de un pequeño dispositivo que, tras conectarlo a la corriente y a nuestro router, comienza a monitorizar todas nuestras conexiones inalámbricas. En el caso de tener algún equipo conectado por cable al router, deberemos conectar el cable directamente a la Whitebox en su lugar; es un sencillo dispositivo plug-and-play que no requiere de mayores conocimientos para comenzar a trabajar y monitorizar la conexión.

El esquema de instalación y cómo trabaja dentro del hogar es el siguiente:

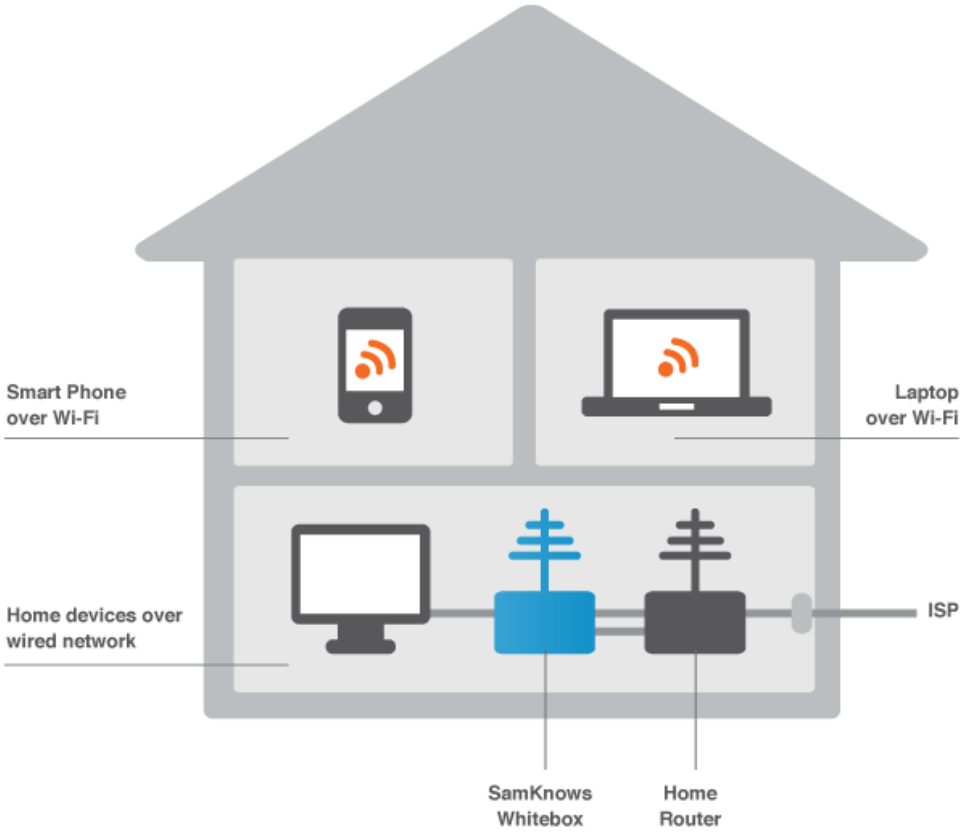


Ilustración 8 – Conexión y modo de trabajo de SamKnows Whitebox [2]

Para obtener las medidas sobre las características de tu red SamKnows realiza diferentes pruebas. En primer lugar, estudia el nivel de tráfico de la red, para realizar sus pruebas en momentos de baja o nula actividad, de manera que los resultados sean más fiables y no interfiera en la navegación del usuario. Las pruebas se realizan de tres maneras diferentes:

- Contra webs populares (Google, Youtube, etc.).
- Nodos de prueba controlados por SamKnows, llamados "Off-net test nodes".
- Nodos de prueba del ISP (Internet Service Provider) del usuario, denominados "On-net test nodes".

Las pruebas que realiza son las siguientes:

1. Prueba de velocidad de descarga multi-roscada HTTP
2. Prueba de velocidad de carga basada en HTTP multi-roscada
3. Disponibilidad de la conexión
4. Jitter
5. Latencia (ICMP y UDP)
6. Pérdida de paquete (ICMP y UDP)
7. Tiempo de resolución de consulta DNS
8. Tasa de fallo de consulta DNS
9. Tiempo de carga de página web
10. Tasa de fallo de carga de página web

Los resultados obtenidos se envían a servidores de SamKnows, desde donde luego se pueden mostrar al usuario de tres maneras diferentes:

- **Reporte en la web**: el usuario puede acceder a su área privada en la web y observar diferentes gráficas en las que se refleja el funcionamiento de su red, con velocidad media de descarga, pérdida de paquetes, latencia, etc. Un gran número de pruebas que permiten dar una imagen global de la conexión del usuario.
- **Reporte mensual**: mensualmente, el equipo de SamKnows envía un informe resumiendo los datos reunidos durante todos los días del mes.
- **Aplicación móvil**: aunque aún no está disponible, próximamente existirá una aplicación para las plataformas iOS y Android donde el usuario podrá consultar en todo momento el comportamiento de su red.

Una imagen global de cómo funcionan las pruebas y el reporte de resultados es la siguiente:

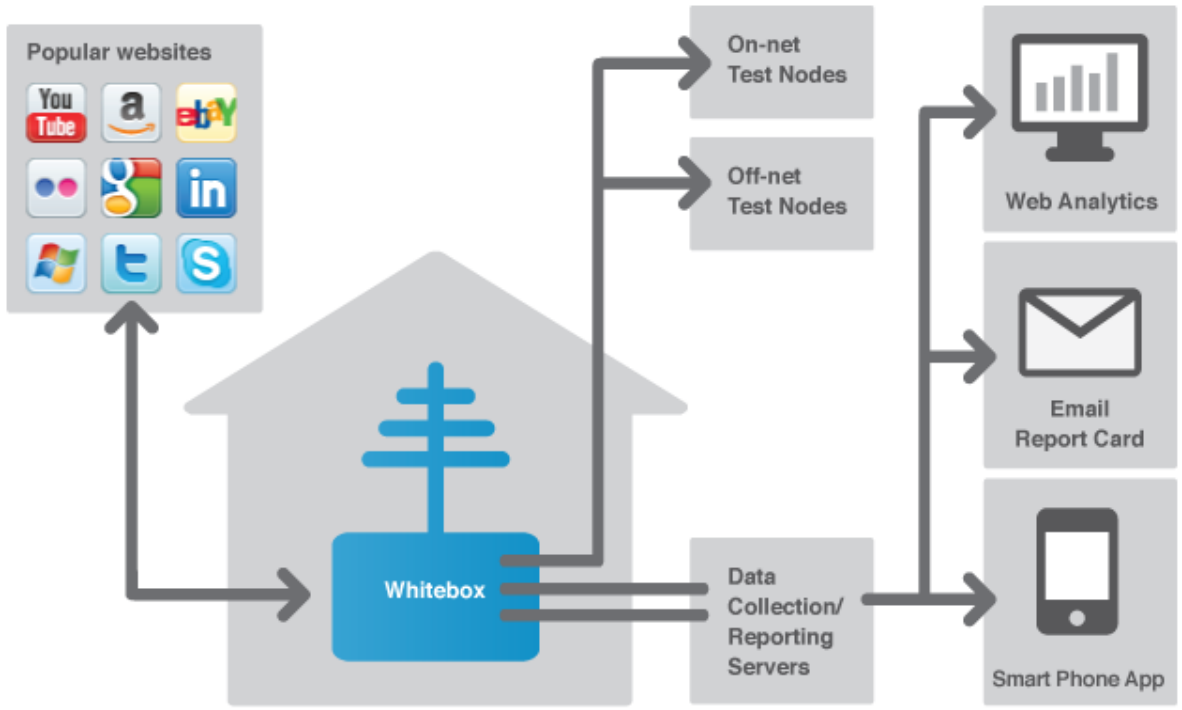


Ilustración 9 – Pruebas y reportes de SamKnows Whitebox [2]

En el caso de que el operador que tenga contratado el usuario cumpla con el código de conducta aprobado por SamKnows y la FCC [25], este podrá acceder a los datos de cada uno de los usuarios, que les serán mostrados de manera anónima. Así, los ISP's podrán adoptar medidas para mejorar sus redes manteniendo la privacidad absoluta del usuario.

2.4 Requisitos y restricciones

En primer lugar, es necesario, obviamente, disponer de una conexión a internet para realizar las comunicaciones. También deberemos tener una Whitebox y acceso físico al router, pues debe ir conectada a éste para realizar sus funciones; por último, se debe contar con algún tipo de cliente SIP, ya sea de videollamadas o de VoIP para poder comunicarse.

2.5 Marco regulador

El trabajo está regulado por las distintas RFC que se usan: RFC 3261 [1], RFC 3550 [7], RFC 3551 [8] y RFC 4566 [3]. Además, estas normativas en ocasiones están regidas por otras normativas del IETF o normas ISO. También cumple con el Código de Conducta de SamKnows registrado en 2012 FCC Broadband Testing and Measurement Program [19].

Capítulo 3

Diseño y desarrollo de la solución

3.1 Introducción

En este apartado abordaremos el diseño y el desarrollo de la solución. En primer lugar, en el diseño, se realiza un análisis de lo que hemos hecho, analizando el porqué de las decisiones tomadas y comparándolo con otras alternativas valoradas.

En segundo lugar, en el desarrollo, explicaremos la solución adoptada, detallando los pasos seguidos y describiendo técnicamente el trabajo realizado.

3.2 Diseño

Hemos decidido utilizar un diseño de tipo cliente-servidor, en el que el cliente será el que tenga a su disposición el hardware específico de Samknows e instalado el programa desarrollado en el proyecto, y por tanto, el que deberá iniciar la comunicación.

Para el desarrollo de la aplicación hemos utilizado tres máquinas virtuales en un servidor, de manera que cada una de ellas cumpliese las funciones de “Cliente”, “Servidor” y “Router”, que a la vez también sería utilizado como “Whitebox”, aunque simulando los datos obtenidos por ésta, pues no los puede obtener de manera real.

El escenario real básico incluiría dos PC a modo de cliente y servidor, un router y una "Whitebox", configurados de manera que el cliente se encontrase en una red y el servidor en otra. Podemos ver el esquema de funcionamiento a continuación:

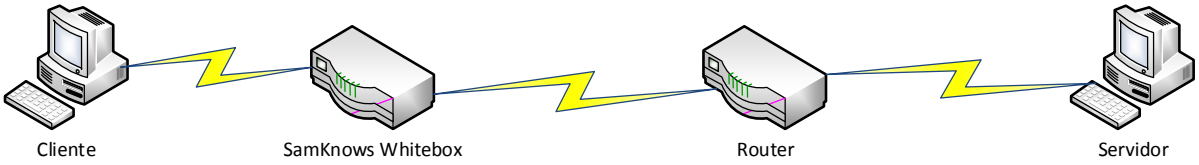


Ilustración 10 – Escenario de funcionamiento básico

Se ha elegido esta forma de trabajo principalmente por comodidad, sencillez y las posibilidades que ofrece. En cuanto al escenario planteado, es el más sencillo que se puede suponer en una comunicación multimedia, y por tanto el más básico para desarrollar una solución inicial. Además, dado que, como veremos más adelante, nuestro programa solo desarrollará una función al realizarse la solicitud INVITE, no tiene sentido complicar más el escenario usando varios receptores, pues los resultados serán similares. Respecto a la decisión de utilizar máquinas virtuales, en lugar de un escenario real, responde a la comodidad de poder trabajar desde cualquier ordenador en cualquier momento, en vez de estar ligado a un sitio físico en el que desarrollar el programa. También tiene el añadido de que si alguna de las máquinas se estropeaba, no supone el coste de trabajo y en ocasiones económico que podría producir una avería en cualquiera de las partes del escenario si éstas fuesen reales. Además, la instalación del software en la WhiteBox no es libre.

Pese a esta comodidad, también ha supuesto un hándicap, debido a la necesidad de crear y configurar esas máquinas virtuales, instalarlas en un servidor de la universidad y obtener el acceso remoto a ellos, lo que en un principio supuso una pérdida de tiempo. Además, al tener que acceder a ellas por SSH, cualquier problema en la conexión impedía acceder a ellas y por tanto al trabajo y a las pruebas. Esto fue muy importante sobre todo en el mes de agosto, donde se apagaron todos los servidores de la universidad, lo que retrasó el trabajo.

A continuación, explicaremos las soluciones tecnológicas elegidas y el por qué, comparándolo con otras opciones en el caso de que las hubiese:

- Se ha elegido SIP como protocolo de negociación de la sesión por ser el protocolo estándar definido en las normativas para establecer comunicaciones multimedia, en la RFC 3261.
- Obviamente, la información multimedia se reflejará en la solicitud INVITE siguiendo las directrices del protocolo SDP, que va asociado a SIP en este tipo de comunicaciones.
- Para el intercambio de información multimedia se ha elegido RTP/RTCP, por tratarse de un protocolo estándar para este tipo de comunicaciones, además de ofrecernos mucha información sobre el estado de éstas.
- El programa se ha desarrollado en Java debido a la gran cantidad de librerías y documentación existentes para el desarrollo de aplicaciones SIP sobre este lenguaje y a la posibilidad de ofrecer, en un futuro, una sencilla instalación en cualquier máquina, sin necesidad de recompilar el programa. También se valoró C como lenguaje de programación, pues nos habría aportado la ventaja de poder trabajar a nivel de Kernel, sin necesidad de subir el paquete a una

aplicación, y por tanto haciendo que el programa fuese aún más rápido. Pese a ello, consideramos que éste no era un factor importante, pues la modificación en Java también se recibe como si se trabajase en tiempo real. Por tanto, el soporte de Java fue determinante para elegirlo finalmente como lenguaje para el proyecto, pues en los aspectos puramente técnicos ambos lenguajes habrían funcionado correctamente.

- El protocolo de transporte elegido para las comunicaciones SIP será UDP, que es el utilizado por defecto. No vamos a cambiarlo pues, para este tipo de comunicaciones en tiempo real, creemos que es más importante que el contenido se reproduzca, aunque falte algún paquete por pérdidas o por llegar tarde, a tener que esperar para que se produzcan las retransmisiones correspondientes como ocurriría con TCP. Es decir, TCP aporta una mayor seguridad y calidad, pero UDP aporta una mayor instantaneidad, algo básico en aplicaciones en tiempo real.
- Utilizaremos el firewall iptables por estar disponible en todos los dispositivos Linux actuales, ya que la mayoría de los routers suelen tener como base este sistema operativo.

Una vez decididos los lenguajes y protocolos que íbamos a usar, y de adquirir los suficientes conocimientos sobre ellos, comenzamos el desarrollo del programa. Su estructura general se muestra a continuación:

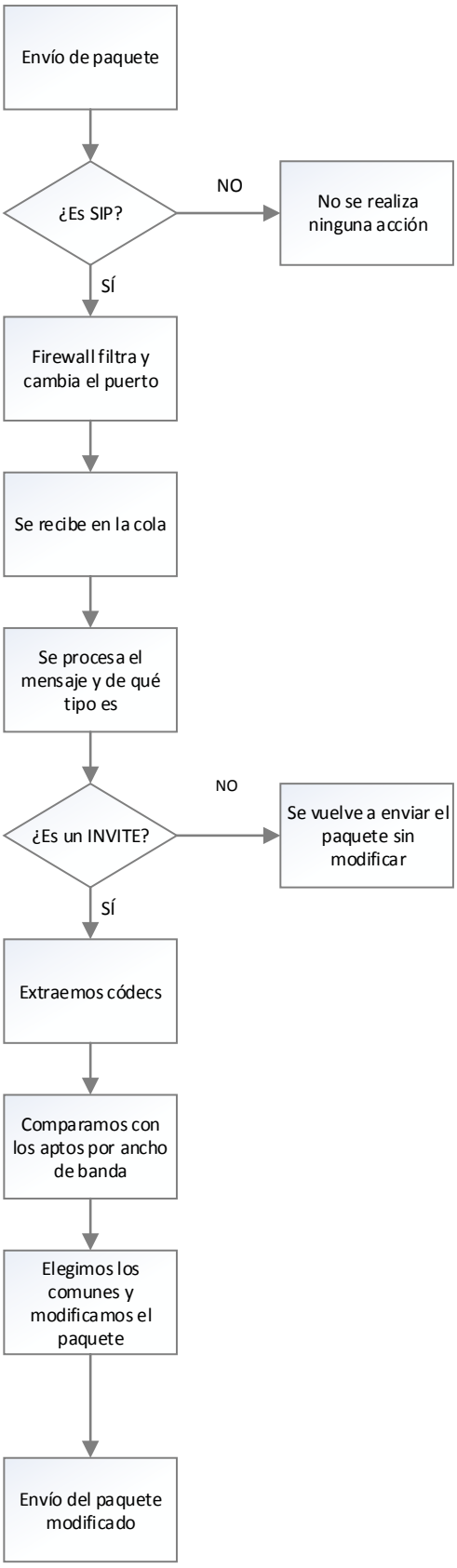


Ilustración 11- Esquema general del programa

La conexión de los diferentes métodos que conforman el programa se muestra a continuación:

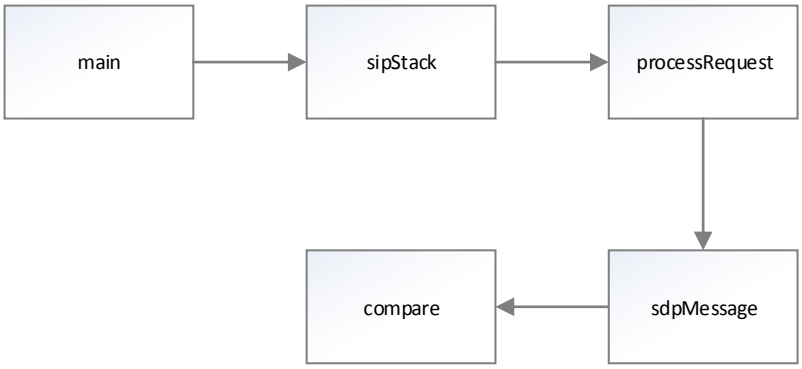


Ilustración 12 – Métodos del programa y su conexión

El resumen de la conexión entre ellos es el siguiente:

1. El método main se encuentra en una clase aparte, donde se hace una llamada al constructor del método sipStack.
2. Este, al recibir un paquete en la pila, llamará al método processRequest pasándole el mensaje recibido como argumento.
3. En este método, se procesará la solicitud y se comprobará de qué tipo es. Si se trata de una solicitud de tipo INVITE, hará una llamada al método sdpMessage con la solicitud como argumento; en caso contrario, reenviará el paquete.
4. Al recibir la solicitud previamente procesada, el método sdpMessage comprobará si contiene carga SDP. En el caso de no haberla, se reenviará el paquete. En cambio, si existe, la procesará y generará un vector con los codecs incluidos en la solicitud, que se utilizarán para llamar al método compare junto con el ancho de banda disponible. Una vez que el método compare devuelve los codecs comunes, modificará el mensaje para reenviarlo al servidor con la nueva información.
5. El método compare seleccionará, en función del ancho de banda, cuáles de los codecs incluidos en la solicitud aportarán una buena calidad, y devolverá un vector con estos datos para que el método sdpMessage pueda modificar el mensaje y reenviarlo.

En el apartado de desarrollo explicaremos claramente cada una de las partes arriba indicadas, cómo y por qué se han realizado así.

3.3 Desarrollo

A continuación iremos describiendo detalladamente cada uno de los pasos que hemos seguido para el desarrollo de este trabajo. Se explicarán cada uno de los pasos de manera que se entienda lo mejor posible el programa globalmente.

3.3.1 Simulación

Lo primero que hemos hecho ha sido instalar el programa "SIPp" [26] que utilizaremos para realizar las simulaciones de las comunicaciones SIP. Se trata de un programa que permite, mediante el uso de dos terminales, representar diferentes escenarios de comunicaciones SIP. El escenario básico consiste en un UAS y un UAC que, una vez ejecutados, comienzan a enviarse paquetes. Por tanto, en primer lugar hemos instalado el programa y probado su funcionalidad más básica, para acostumbrarnos a su funcionamiento y comprobar que la conexión se realizaba correctamente.

Además de estos dos escenarios y otros que incluye el programa y que aportan otras características más o menos complejas, el usuario puede generar sus propios escenarios para que funcionen de la manera que el desee. Los escenarios se generan como archivos XML y en ellos el usuario puede añadir las características que se adecúen mejor a sus necesidades. En nuestro caso, modificamos el cliente proporcionado para añadir más codecs en el apartado multimedia, para así después poder comprobar el funcionamiento del método que compara entre éstos y los aptos en función del ancho de banda existente en ese momento.

3.3.2 Filtrado

A continuación lo que debemos hacer es filtrar los paquetes de nuestro interés, de manera que después podamos utilizarlos y modificarlos. Para ello, haremos uso, como ya se indicó, de iptables, el firewall incluido en todas las distribuciones de Linux actuales.

Dado que el servidor con el que queremos contactar puede estar en cualquier dirección IP, ésta no sería un buen criterio para filtrar los paquetes, pues no sabríamos cuándo se trata de un paquete SIP o no. En cambio, los paquetes SIP van dirigidos, por defecto, al puerto 5060, por lo que esta información sí que nos será útil para seleccionar los paquetes deseados.

Sabiendo esto, hemos realizado un filtro que redirige los paquetes con destino al puerto 5060 sobre el protocolo de transporte UDP hacia el puerto 5080, puerto en el que estaremos esperando el paquete. De esta manera, el paquete de nuestro interés irá hacia nuestro programa antes de llegar a su destino, y podremos modificarlo sin que éste se dé cuenta. Hemos elegido el puerto 5080 sin un propósito concreto, simplemente por tratarse de un puerto libre. Podría haber sido cualquier otro que no estuviese reservado para alguna función concreta.

Este filtro no afecta al paquete, solo hace un puente virtual entre ambos puertos. De esta forma, una vez modificado el mensaje, cuando se reenvíe el servidor, el puerto de destino seguirá siendo el 5060, el habitual en comunicaciones SIP.

Para realizar este cambio usando iptables utilizaremos el siguiente comando:

```
iptables –t nat –A PREROUTING –p udp –dport 5060 –j REDIRECT –to-port 5080
```

De esta manera, conseguimos el objetivo de desviar los paquetes de una manera sencilla e integrada en todos los dispositivos Linux actuales.

3.3.3 Recepción del paquete

Una vez hemos desviado el paquete al puerto de destino deseado, deberemos tener a nuestro programa "escuchando" en él, para que lo pueda recibir y así después trabajar sobre él. Para ello, en primer lugar creamos una pila ("sipstack") que será la que reciba los paquetes cuando lleguen. A estas pilas se les pueden agregar diferentes propiedades para que realicen ciertas funciones, como por ejemplo depuración, pero en nuestro caso únicamente le asignaremos un nombre, en este caso "myStack". Para las tareas que realizaremos en el proyecto, no es necesario añadirle más propiedades, simplemente que reciba los paquetes.

Para que los paquetes lleguen a la pila, se debe tener un escuchador ("listening point") en el puerto y dirección IP deseados. Como ya comentamos, hemos elegido el 5080 por tratarse de un puerto libre, pero podríamos haber elegido cualquier otro que no tuviese ninguna función específica, y como dirección IP la 192.168.1.2, que es la dirección de nuestro servidor, al que irán dirigidos los paquetes. Es importante que la dirección IP sea la del servidor de destino, pues sino, aunque estuviésemos en el puerto correcto, no recibiríamos nada al no escuchar en la dirección de destino. Se tratará de un escuchador para paquetes sobre el protocolo de transporte UDP, pues como dijimos anteriormente, será el protocolo habitual en las comunicaciones SIP, y en nuestro caso, el usado siempre.

3.3.4 Procesado del paquete

Cuando recibimos un paquete en la pila, lo primero que se debe realizar es extraer la información del paquete, la solicitud ("request") en sí misma. Para ello, utilizaremos varios métodos propios de las librerías JSIP, que permiten extraer la solicitud y el tipo de ésta. Ya que debemos de saber de cuál se trata en cada caso para poder trabajar con él.

Una vez tenemos la información de la solicitud, es sencillo obtener de qué tipo es. Por último, dado que a nosotros solo nos interesan las solicitudes de tipo INVITE, comprobaremos si el paquete en cuestión es de ese tipo, en cuyo caso procederemos a seguir trabajando con él, y sino simplemente lo reenviaremos hacia el servidor mediante el método "sendRequest" de JSIP.

En el caso de tratarse de un INVITE, procederemos a trabajar con el mensaje para obtener la información deseada y posteriormente seleccionar los codecs óptimos para las características de nuestra red en ese momento. Para ello, en primer lugar, realizaremos una llamada al método sdpMessage, que realizará las funciones necesarias para obtener la información SDP y modificar el paquete con la nueva, como veremos a continuación, tras el esquema que contiene los pasos generales para esta modificación:

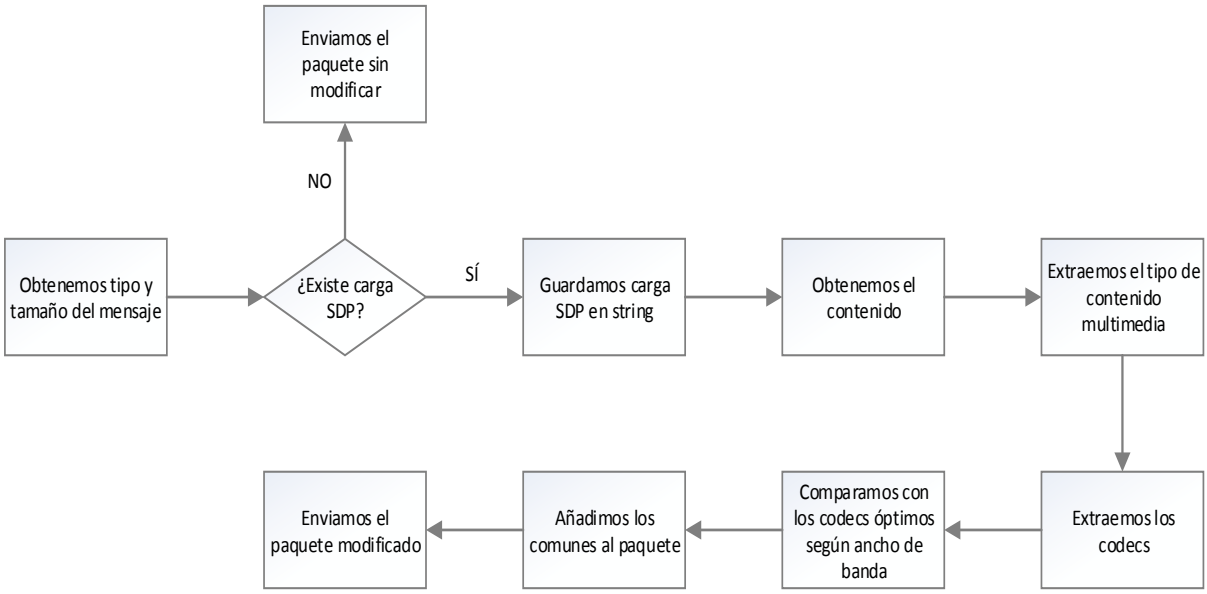


Ilustración 13 – Procesado general de solicitud INVITE

En primer lugar, debemos saber el tipo y tamaño del paquete, que obtendremos de las cabeceras. Esto no es básico para el desarrollo del programa, pero es importante para poder comprobar que el paquete contiene información (longitud>0) y que es de tipo SDP, pues sino no nos interesa y podría hacer fallar al programa en los siguientes pasos, pues no existiría información SDP con la que trabajar.

En el caso de que el paquete no contenga información SDP, se reenviará hacia el servidor como en el anterior, sin realizar ninguna modificación en él. En el caso contrario, trabajaremos con la solicitud para extraer de ella la información multimedia que nos permitirá elegir los mejores codecs en cada momento.

El esquema detallado de cómo se gestiona la carga SDP se muestra a continuación:

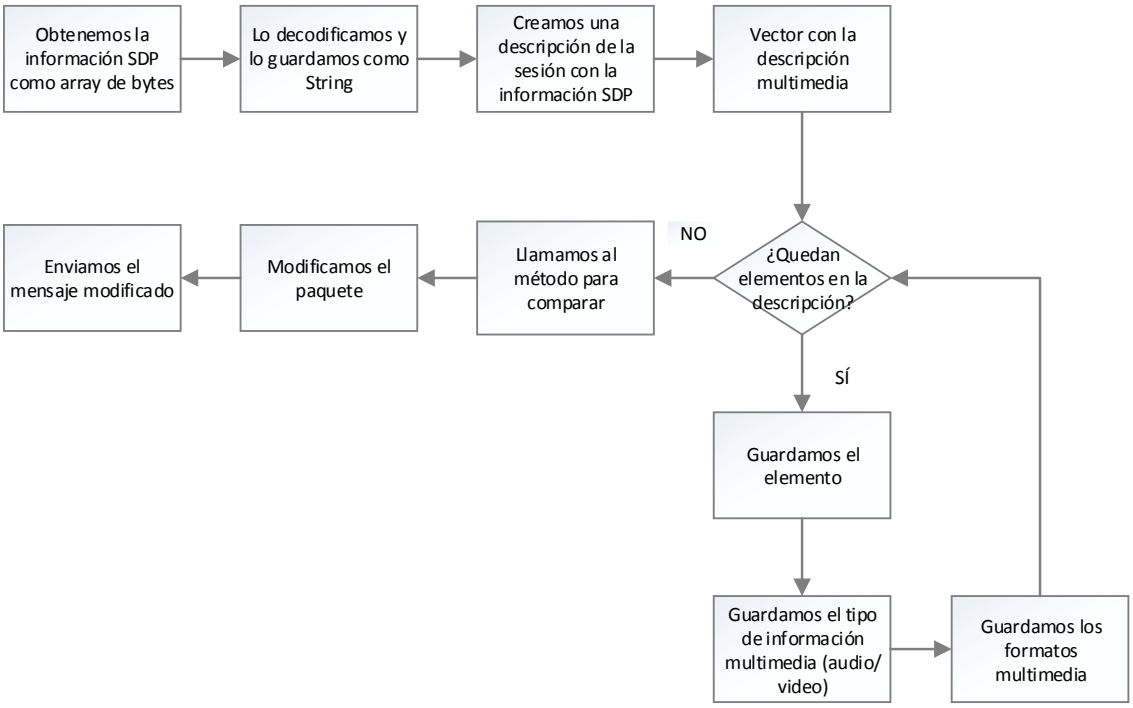


Ilustración 14 – Esquema detallado del procesado SDP

Lo primero que haremos será guardar la información del mensaje en un string. No podemos obtenerla directamente, sino que primero deberemos guardar la información como un array de bytes, en formato plano, para después, mediante el uso de un sistema de decodificación (en este caso deberá ser "UTF-8" [1]), transformar la información simple en un string que contenga la carga SDP decodificada. Es importante hacer este cambio para después poder parsear esa información y guardar los datos necesarios para nuestro programa, pues no podemos hacerlo a partir de los datos representados como bytes.

Una vez hemos guardado en un string el contenido SDP, lo primero que debemos hacer para poder guardar los formatos enviados en la solicitud INVITE es crear una descripción de la sesión multimedia con la información del contenido SDP al completo. A partir de ahí, obtendremos toda la información multimedia, para posteriormente ir obteniendo información más concreta hasta conseguir el vector con los formatos incluidos en la solicitud.

En primer lugar, obtendremos el tipo de contenido multimedia que se va a describir, es decir, si se trata de audio o vídeo, de manera que después podamos tomar las decisiones en base a eso. También sabremos el protocolo utilizado para el intercambio de mensajes una vez iniciada la comunicación (RTP/AVP). Tras eso, mediante el método getMediaFormats, obtendremos un vector que contendrá los codecs incluidos en la solicitud. Esto nos va a generar un problema, pues los formatos nos los guarda por el valor asociado a cada uno de ellos, y no por su nombre. Por ejemplo, en el caso de que la información incluida en la solicitud INVITE fuese:

```
m=audio [media_port] RTP/AVP 0 3
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
```

El vector que obtendríamos sería:

```
Formatos=[0,3]
```

El problema reside en que, como se muestra en la tabla correspondiente a los formatos de RTP, algunos de los codecs de audio no tienen un número fijo, sino que se les asigna uno dinámicamente; esto implica que no podremos realizar la comparación con ellos, pues no podemos asociar el códec correspondiente. Por tanto, estos codecs deberán ser eliminados directamente, pues nunca coincidirán con ninguno de los propuestos en las tablas de comparación.

Ahora que ya tenemos un vector con los formatos incluidos en la solicitud INVITE, llamaremos al método que realiza la comparación entre los formatos incluidos en ese vector y los que nos darán una buena calidad con el ancho de banda disponible. Este método se basa en una serie de condicionales "if-else" en función del ancho de banda; dependiendo éste, añadiremos unos formatos u otros al vector donde reuniremos los válidos para las condiciones de ese momento. Dado que el bitrate de los codecs no siempre es exacto, y que la red sufre pequeñas variaciones de capacidad continuamente, hemos decidido generar únicamente cinco grupos para el caso del audio, y siempre dejando un cierto margen por encima del máximo bitrate del grupo, de manera que aunque la red fluctúe ligeramente, no se reproduzcan los problemas típicos de estas comunicaciones que estamos tratando de evitar con este programa. Los diferentes grupos los recoge la siguiente tabla:

Ancho de banda disponible (kb/s)	**Codecs válidos**
BW<10	Ninguno
10<=BW<25	7 (LPC)
	4 (G723)
	18 (G729)
	12 (QCELP)
25<=BW<45	3 (GSM)
	15 (G728)
45<=BW<75	14 (MPA)
	5 (DVI4)
	6 (DVI4)

	16 (DVI4)
	17 (DVI4)
75<=BW<1400	0 (PCMU)
	8 (PCMA)
	9 (G722)
BW>=1400	10 (L16)
	11 (L16)

Tabla 6 - Codecs de audio disponibes en funcion del ancho de banda

Obviamente, en cada grupo estarán incluidos los codecs de ese grupo y los de todos los anteriores que tienen unos requisitos inferiores.

En el caso de vídeo, dado que las diferencias de bitrate son mayores, las seleccionamos individualmente. También en este caso dejaremos cierto margen en el ancho de banda requerido, aunque en este caso los valores ya se tratan de una estimación y no de algo seguro. La selección se realiza en base a la siguiente tabla:

Ancho de banda disponible (kb/s)	Codecs válidos
BW<1500	Ninguno
1500<BW<=2000	32 (MPV)
2000<BW<=4200	31 (H261)
4200<BW<=6000	34 (H263)
6000<BW<=7200	28 (nv)
BW>7200	26 (JPEG)

Tabla 7 - Codecs de vídeo disponibes en funcion del ancho de banda

Una vez tenemos guardado en un vector los codecs de audio o vídeo que funcionarán correctamente con el ancho de banda, comparamos éste con el vector de los formatos disponibles en la solicitud INVITE, y elegimos los comunes. De esta manera, tendremos un vector en el que se incluirán los codecs que se adaptarán tanto a las características del cliente como al ancho de banda de su red en el momento de la solicitud.

A continuación, deberemos modificar la información multimedia del paquete para añadirle la que acabamos de calcular. En un primer momento, pensamos que, al llamar al método setMediaFormats con el vector que contiene los formatos comunes, se modificaría la información multimedia obteniendo el resultado final deseado. Sin embargo, al probarlo, nos dimos cuenta de que este método solo modificaba los valores en la línea de la información multimedia (m=), pero no los atributos de cada códec (a=). Por tanto, el siguiente diagrama muestra el proceso seguido finalmente:

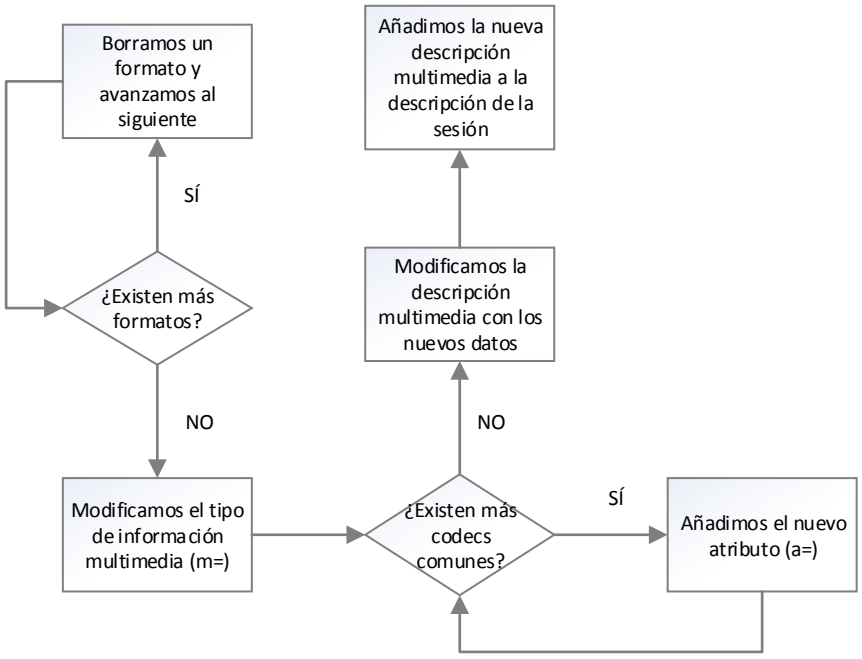


Ilustración 15 - Modificación carga SDP

Como se desprende del diagrama, tenemos que realizar la acción en dos pasos:

- **Eliminar la información existente:** en primer lugar, deberemos borrar la información multimedia incluida originalmente en el paquete. Para ello, debemos ir eliminando cada uno de los atributos existentes, por lo que recurrimos a un bucle "for" en el que, en cada iteración, se elimina un atributo, mientras que sigan quedando formatos por borrar. Eliminaremos únicamente atributos del tipo "rtpmap", que son los que incluyen la información multimedia. En el caso de existir otros (type, charset…), no serán modificados.
- **Añadir los nuevos formatos:** antes de añadir los codecs que hemos seleccionado para la solicitud, modificaremos la línea en la que se indica el tipo de actividad multimedia (audio/vídeo) y los codecs que se incluirán, de manera que aparezcan los que hemos decidido previamente. A continuación, para añadir los codecs que incluiremos en la nueva descripción multimedia, deberemos llamar al método "addDynamicPayloads" con dos vectores, el primero de los cuales se corresponderá con el tipo de atributo (en nuestro caso, "rtpmap" en todas las ocasiones) y el segundo con la información que debe contener (por ejemplo, "0 PCMU/8000"). Ambos vectores deben tener las mismas dimensiones. El problema es que, en nuestro caso, no sabemos qué tamaño han de tener, pues depende del número de codecs comunes entre la solicitud INVITE y las características de la red. Para solucionarlo, hemos recurrido a un bucle "for" en el que, en cada iteración, se añade un elemento "rtpmap" al vector del tipo de atributo, y la información correspondiente en el otro vector, mientras queden más codecs en común.
  Otro problema es que, en principio, solo sabemos el número de los codecs comunes, pero no el nombre del códec correspondiente. Como solución, creamos un array en el que se incluyen los nombres de cada códec en la posición correspondiente a su número. De esta manera, podemos relacionar

ambos datos y así generar la información completa que posteriormente añadiremos. Esta información será [valid.elementAt(posicion)+ " " +posibles[posicion]+"/8000"] en el caso de audio (para vídeo, será /90000). Así, podremos añadir todos los codecs disponibles tanto de audio como de vídeo, de manera totalmente automática.

Por último, una vez hemos modificado la carga SDP, debemos aplicar estos cambios al mensaje general, para que al enviar éste al servidor llegue con la información modificada. Para ello, como en el caso anterior, primero deberemos eliminar la descripción de sesión ya incluida, para posteriormente incluir la nueva junto con el tipo de contenido, que en este caso no cambiará, pues seguirá siendo SDP. A continuación, enviaremos el mensaje modificado haciendo uso del método “sendRequest”, al que le pasaremos como parámetro el mensaje actualizado.

Tras este proceso, el mensaje se recibirá correctamente en el servidor, que podrá continuar con la negociación de la sesión, pero en base a unos codecs actualizados para obtener una calidad óptima.

Capítulo 4

Pruebas, resultados y evaluación

4.1 Introducción

En los siguientes dos apartados explicaremos, en primer lugar, las pruebas realizadas para comprobar el funcionamiento del programa, explicando por qué se han elegido esas y mostrando los resultados obtenidos.

En el segundo apartado, analizaremos los resultados obtenidos anteriormente, comentando las ventajas e inconvenientes del programa y comparándolos con los resultados esperados, con el fin de analizar el trabajo en su totalidad.

4.2 Pruebas

Las pruebas se han realizado utilizando las máquinas virtuales instaladas y el programa SIPp para establecer las comunicaciones. Nuestro programa se ejecutará en la máquina correspondiente al router, mientras que en las otras dos tendremos el cliente y el servidor. Utilizaremos el escenario de cliente modificado para incluir los codecs adecuados, mientras que para el servidor utilizaremos el proporcionado por SIPp sin ningún tipo de modificación.

A continuación, explicaremos las diferentes pruebas llevadas a cabo para comprobar el funcionamiento del programa. Como ya se ha indicado, estas se realizarán de forma remota en las máquinas virtuales, simulando tráfico real entre ellas.

1- **Pruebas de audio:** en primer lugar, realizamos pruebas para comprobar el correcto funcionamiento cuando solo se incluían codecs de audio. Para ello, en el cliente, añadimos un códec de cada grupo de ancho de banda de audio, y fuimos modificando el valor de éste para comprobar que seleccionaba los correctos. De esta manera, pretendemos demostrar el funcionamiento del programa, tanto en la modificación del mensaje como en su posterior reenvío. Solo se envía una solicitud entre cliente y servidor. Los resultados se muestran en las siguientes imágenes, siendo el terminal de la izquierda el correspondiente al cliente y el de la derecha al servidor:

- **Ancho de banda 15Kbps:**

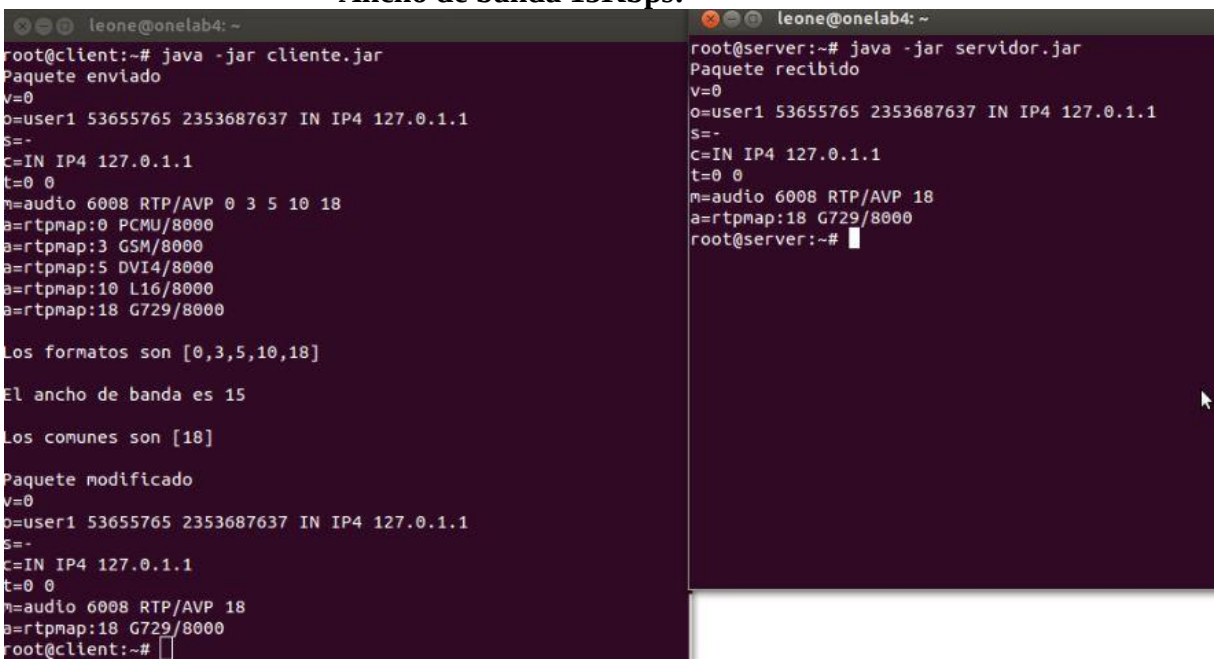


Ilustración 16 – Caso 1: ancho de banda entre 10 y 25 Kbps

- **Ancho de banda 30Kbps:**

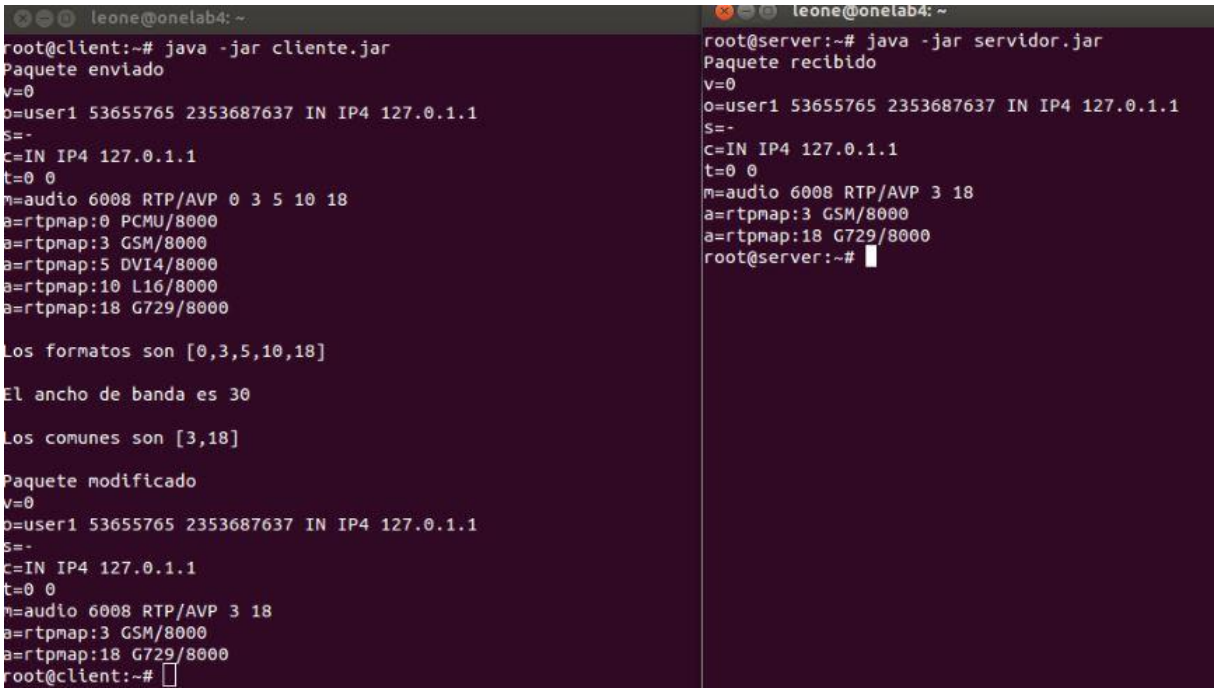


Ilustración 17 – Caso 2: ancho de banda entre 25 y 45 Kbps

- **Ancho de banda 60Kbps:**

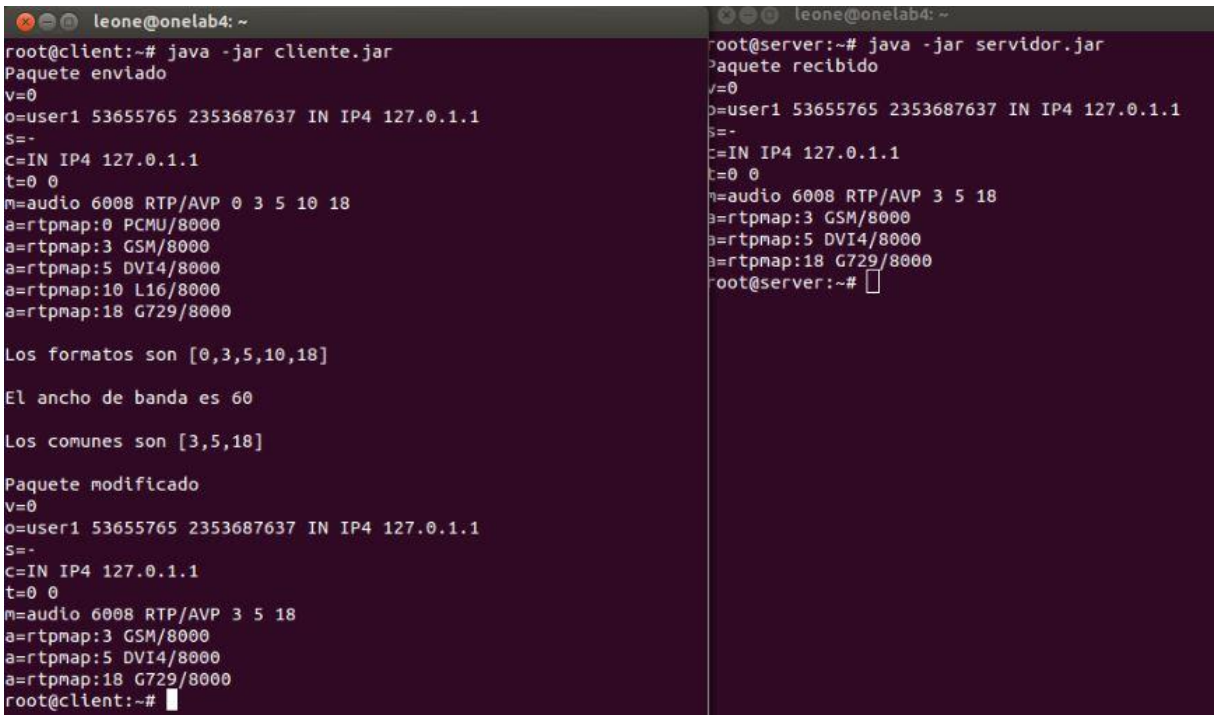


Ilustración 18 - Caso 3: ancho de banda entre 45 y 75 Kbps

- **Ancho de banda 100 Kbps:**

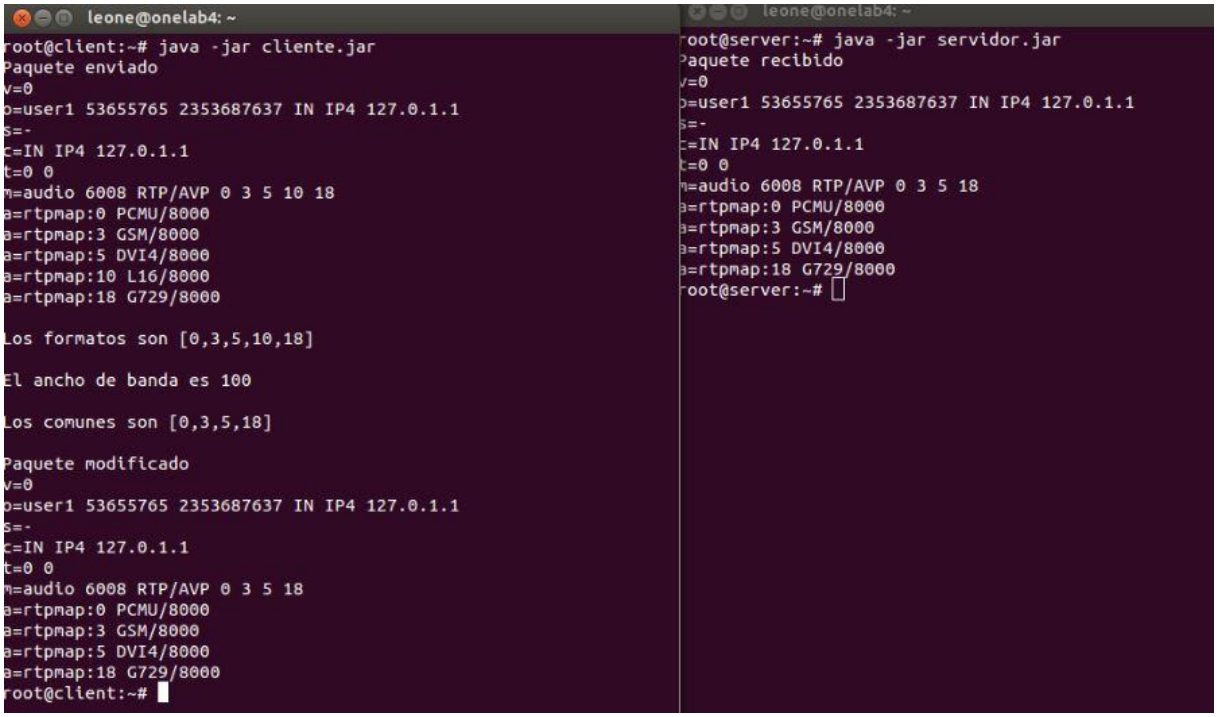


Ilustración 19 – Caso 4: ancho de banda entre 75 y 1400 Kbps

- **Ancho de banda 1500 Kbps:**

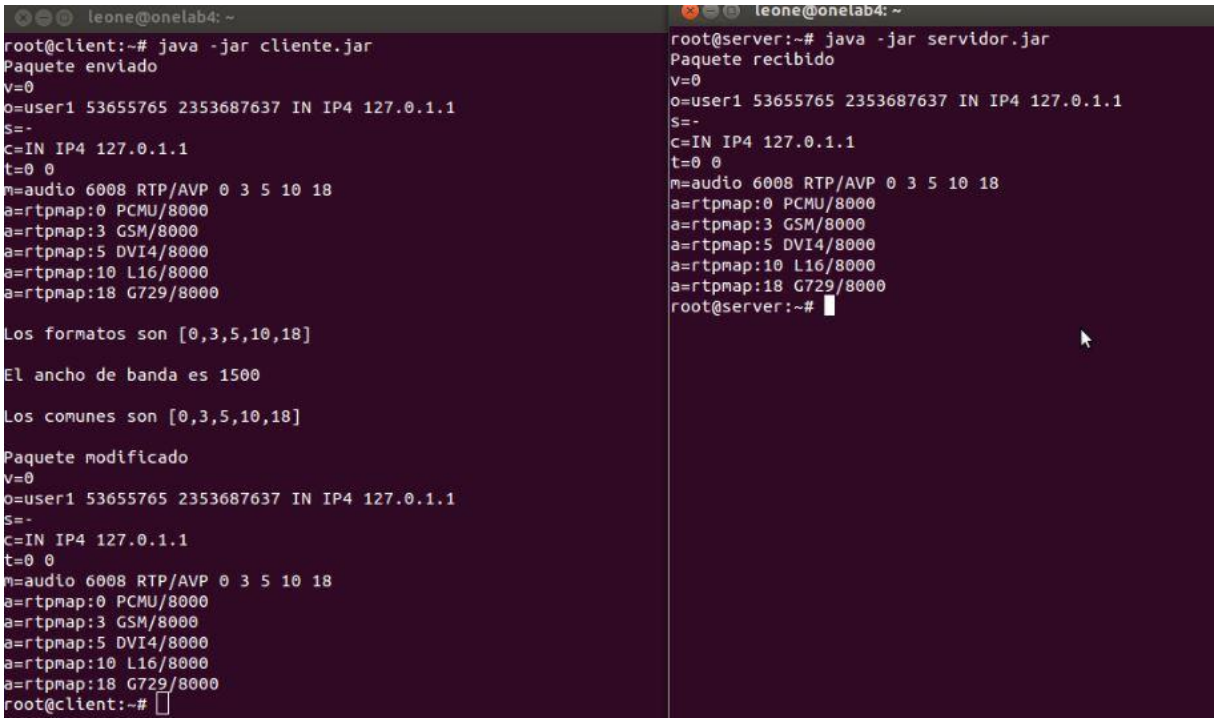


Ilustración 20 – Caso 5: ancho de banda 1500 Kbps

2- **Pruebas de vídeo:** en segundo lugar, se realizaron pruebas para comprobar la funcionalidad en el caso del vídeo. Para ello, modificamos una vez más el escenario del cliente, añadiendo la información de vídeo en lugar de la de audio. Se añadieron todos los formatos y se fue modificando el ancho de banda para comprobar la correcta selección de codecs en cada caso.Para no repetir tantos casos de cara a la memoria, mostraremos una única prueba, en la que se selecciona entre dos codecs. En este caso también se envía un único paquete entre el cliente y el servidor. El resultado se muestra a continuación:

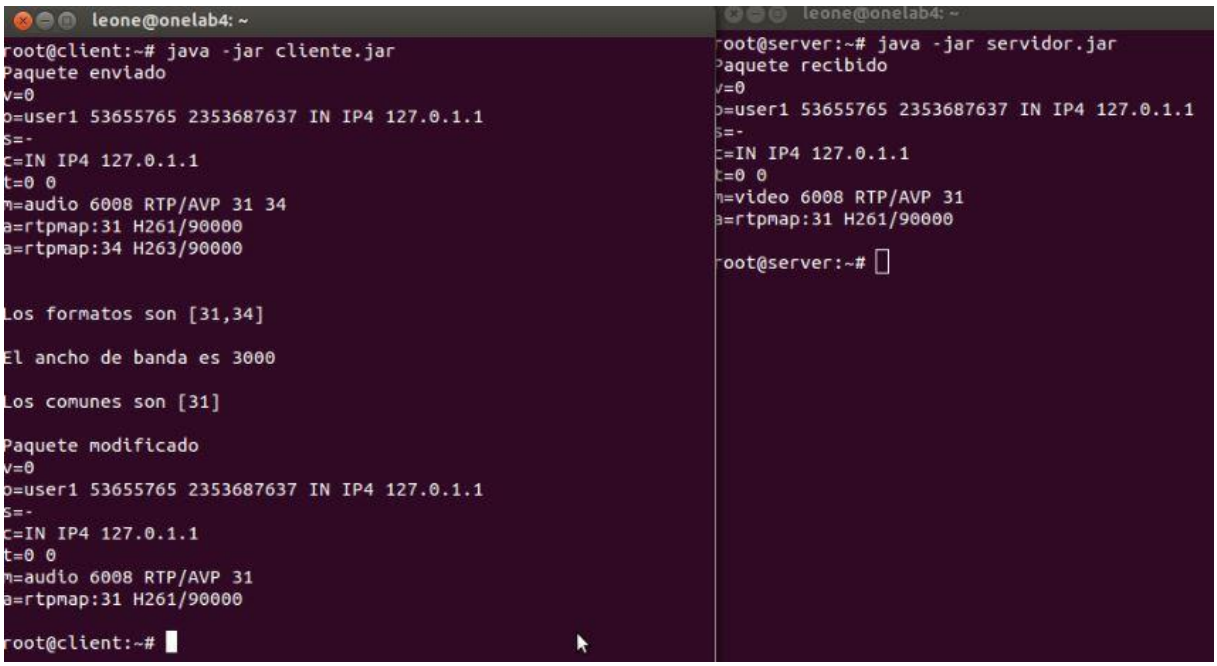


Ilustración 21 – Prueba de vídeo, con ancho de banda entre 2000 y 6000 Kbps

3- **Pruebas de audio y vídeo combinado:** en esta última combinación, se añadieron a la solicitud tanto codecs de audio como de vídeo. Esto ocasiona que el programa dé un error al obtener la información multimedia y no funcione correctamente.

4- **Pruebas con envío de varios paquetes:** por último, se modificó el cliente para que enviase solicitudes periódicamente. El programa procesa perfectamente las treinta primeras, dejando de recibirlas después y por tanto de reenviarlas al servidor. En cuanto se detiene y se vuelve a ejecutar el servidor, puede recibir otros treinta paquetes sin problemas. Se puede ver el problema en la siguiente imagen:

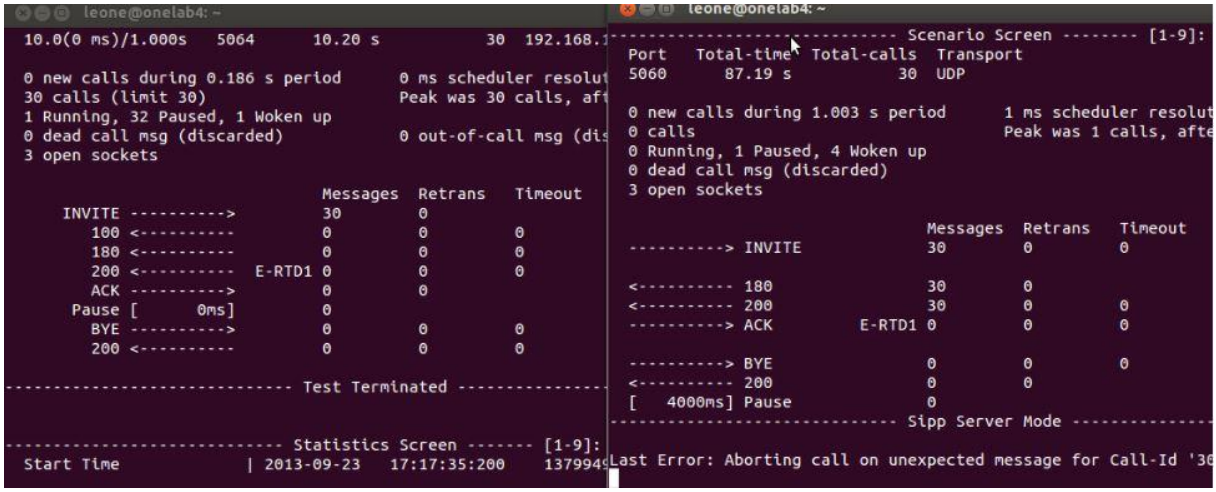


Ilustración 22 – Fallo en el envío al llegar a 30 mensajes

4.3 Análisis de resultados y evaluación

Los resultados obtenidos son, en la mayoría de los casos son los esperados, pues tanto para los casos de audio como de vídeo selecciona perfectamente los codecs adecuados en cada momento, modificando correctamente el mensaje y recibiéndose de manera satisfactoria en el servidor. Esto producirá una mejora en los casos en los que el ancho de banda disponible sea menor que el bitrate requerido por el códec, y por tanto el usuario recibirá una mejor calidad de servicio, con un MOS superior.

Por el contrario, existen dos casos en los que el programa no cumple con su función. En el caso de la combinación de audio y vídeo, supone un hándicap, pues en el caso de videoconferencias muchas veces se especifica un códec de audio y otro de vídeo, aunque en otros casos el vídeo lleva el audio embebido, lo que solucionaría el problema. Tratamos insistentemente de solucionar el problema con variaciones en el código, pero no conseguimos una solución final. Por otra parte, el problema al llegar a las treinta solicitudes no es tan grave, pues generalmente no se realizan tantas solicitudes seguidas. De todas formas, tratamos de buscar una solución, pero no hallamos ninguna. La pila donde se reciben los paquetes no está definida con un número máximo de treinta, por lo que no alcanzamos a ver cuál podría ser el problema.

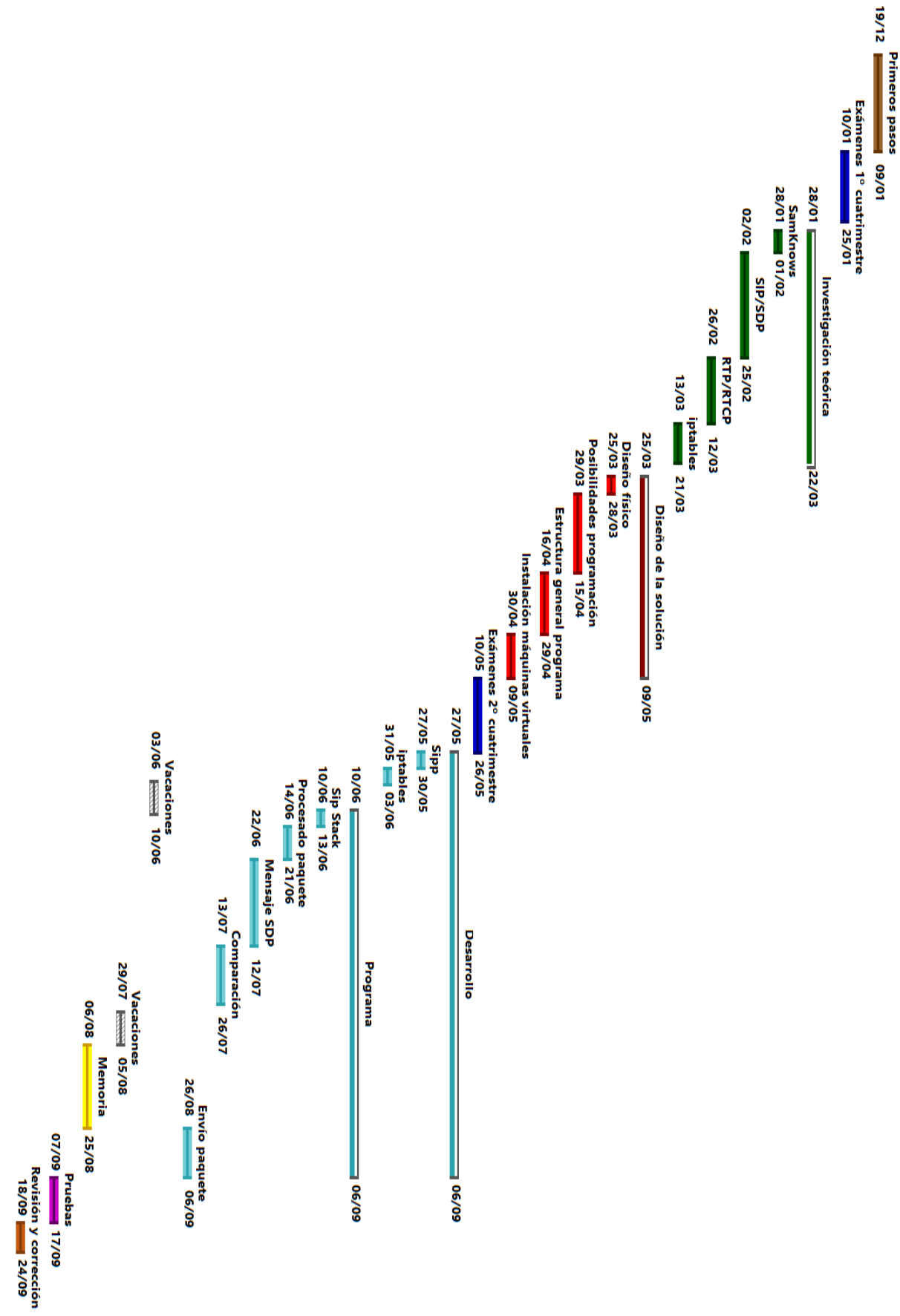
Por tanto, podríamos decir que el programa realiza correctamente sus funciones básicas, pero genera fallos en ciertos entornos. De todas maneras, supone una mejora en las comunicaciones en la mayoría de los casos.

Capítulo 5

Planificación y presupuesto

5.1 Planificación

En este apartado trataremos de explicar cómo se ha planificado este proyecto y se ha estructurado el trabajo, desde el momento en el que fue adjudicado hasta su entrega. Para ello, a continuación se muestra un diagrama de Gantt que posteriormente explicaremos:


Primeros pasos
19/12
09/01
Exámenes 1° cuatrimestre
10/01
25/01
Investigación teórica
28/01
22/03
SamKnows
28/01
01/02
SIP/SDP
02/02
25/02
RTP/RTCP
26/02
12/03
iptables
13/03
21/03
Diseño de la solución
25/03
09/05
Diseño físico
25/03
28/03
Posibilidades programación
29/03
15/04
Estructura general programa
16/04
29/04
Instalación máquinas virtuales
30/04
09/05
Exámenes 2° cuatrimestre
10/05
26/05
Desarrollo
27/05
06/09
Sipp
27/05
30/05
iptables
31/05
03/06
Programa
10/06
06/09
Sip Stack
10/06
13/06
Procesado paquete
14/06
21/06
Mensaje SDP
22/06
12/07
Comparación
13/07
26/07
Vacaciones
03/06
10/06
Vacaciones
29/07
05/08
Memoria
06/08
25/08
Envío paquete
26/08
06/09
Pruebas
07/09
17/09
Revisión y corrección
18/09
24/09

5.1.1 Primeros pasos

Tras la adjudicación del proyecto, y hasta el comienzo de los exámenes del primer cuatrimestre, aprovechamos el tiempo para investigar sobre la situación general de las comunicaciones multimedia, los problemas existentes en ellas y las posibles soluciones, además de organizar el enfoque general del proyecto y las líneas de trabajo.

5.1.2 Investigación teórica

Tras esta primera fase dedicada a situar un poco el proyecto en su contexto y finalizar los exámenes, periodo en el que el proyecto quedó en un segundo plano, iniciamos la investigación en profundidad de los diferentes aspectos teóricos del proyecto, con el objetivo de tener el máximo de conocimientos que nos permitiesen afrontar la parte práctica de la mejor forma posible.

En primer lugar, recabamos información sobre SamKnows y su WhiteBox, ya que no teníamos ninguna información al respecto y era uno de los puntos base del proyecto, enmarcado dentro del proyecto europeo llevado por esta empresa.

Tras ello continuamos con los protocolos SIP y SDP, pues era el punto fuerte para la parte técnica del proyecto. Toda la parte de modificación de los paquetes iba a estar basada en estos dos protocolos, por lo tanto era necesario conocer en profundidad tanto su funcionamiento como su estructura, de manera que después pudiésemos utilizar esa información para desarrollar el mejor diseño posible.

Seguimos la formación teórica con RTP y RTCP, los protocolos elegidos para el intercambio de paquetes multimedia. Esto también era muy importante, pues de ellos extraemos los codecs soportados para la comunicación multimedia, y también porque era necesario para alguna de las ideas que se podrían desarrollar más adelante. Aunque finalmente no se utilizaron, se pueden ver reflejadas en las posibles líneas de trabajo futuras.

Por último, estudiamos el funcionamiento de iptables y sus posibilidades como firewall, de manera que después pudiésemos elegir la mejor opción de diseño, como ya se comentó en el apartado correspondiente.

5.1.3 Diseño de la solución

Tras adquirir los conocimientos teóricos necesarios para iniciar la parte técnica, comenzamos a diseñar la solución que posteriormente desarrollaríamos.

En primer lugar, nos centramos en la decisión sobre cómo construir el escenario, tanto en un entorno real, como en uno simulado, de manera que se pudiese trabajar remotamente y con seguridad durante el desarrollo del programa.

A continuación, valoramos las diferentes posibilidades de programación, los diferentes lenguajes y sus posibilidades, y en función de la decisión tomada, también optar por un tipo

de firewall u otro con iptables. También decidimos qué programa utilizar para simular las comunicaciones multimedia que el programa deberá gestionar.

Seguimos haciendo un esquema general del programa, de manera global y como borrador, para tener una visión de los diferentes métodos necesarios, cómo se conectan las diferentes partes entre ellas y en qué orden trabajar en el desarrollo.

Por último, y tras decidir las máquinas necesarias para el desarrollo del proyecto, solicitamos su instalación en la universidad y el acceso público de manera remota a ellas para acceder desde cualquier ordenador con conexión a internet.

5.1.4 Desarrollo

Tras un parón para preparar y hacer los exámenes finales del segundo cuatrimestre, iniciamos el desarrollo del programa en sí, tras haber adquirido la formación teórica necesaria y haber realizado el diseño de la solución, comenzamos con su desarrollo.

Para empezar, trabajamos sobre el programa SIPp y sus posibilidades de funcionamiento, para posteriormente crear nuestro propio escenario adaptado a nuestras necesidades.

A continuación, procedimos a crear el filtro deseado mediante el uso de iptables, de manera que los paquetes ya se redirigiesen al puerto deseado, y comprobamos el correcto funcionamiento del filtro con el programa SIPp.

Una vez que la estructura para las pruebas estaba creada con la preparación del programa SIPp y del firewall iptables, comenzamos con el desarrollo del programa. Lo primero fue crear la pila en la que se recibirían los paquetes, de manera que después pudiésemos comenzar con su tratamiento.

Continuamos trabajando sobre el tratamiento del paquete, en primer lugar obteniendo su tipo para posteriormente, en función de éste, analizarlo con el fin de extraer la carga SDP de él para poder utilizarla en los siguientes pasos.

Una vez obtenida esa información, nos centramos en el método para comparar los codecs incluidos en la solicitud INVITE con los aptos con el ancho de banda disponible en ese momento, de manera que eligiésemos la mejor opción.

Por último, creamos el método para actualizar el paquete con la nueva carga SDP y posteriormente enviarlo al servidor, de manera que este procese la solicitud y se inicie la comunicación multimedia.

5.1.5 Memoria

La memoria se fue realizando a lo largo de todo el proceso, añadiendo información, referencias y esquemas durante cada uno de los pasos desarrollados en el proyecto; de todas maneras, durante gran parte del mes de agosto aprovechamos para redactar correctamente la memoria, sobre todo lo referente a la parte teórica, pues no podíamos trabajar en el desarrollo

del programa al estar apagados los servidores de la universidad, y por tanto no disponer de un banco de pruebas para las modificaciones hechas.

5.1.6 Pruebas

Una vez completada la primera versión del programa, fue el momento de realizar diferentes pruebas en profundidad, para comprobar su correcto funcionamiento y extraer conclusiones sobre los resultados obtenidos.

5.1.7 Revisión y corrección

Por último, ya con el programa y la memoria completados, nos dedicamos a revisar y corregir ambas partes, pero sobre todo la memoria, en busca de fallos gramaticales y de redacción, de estilo, pies de foto y de tabla, etc., de manera que la versión entregada fuese lo más correcta posible.

5.2 Presupuesto

A continuación detallaremos el presupuesto necesario para la realización de este proyecto:

5.2.1 Coste de personal

Coste por las horas empleadas por el tutor y/o otros profesores:

- Horas tutor (aprox.): 3h/semana x 35 semanas x 15€/h
- Otros profesores (aprox.): 20h x 15€/h

5.2.2 Coste de material

Existen ciertos costes por los materiales utilizados:

- Ordenador portátil: 599€
- Licencia Office Home&Students: 119€
- Conexión ADSL: 50€/mes x 9 meses

5.2.3 Resumen de costes

La siguiente tabla resume los costes totales del proyecto:

Costes de personal	
Tutor	1575€
Otros profesores	300€
TOTAL	**1875€**
Costes de material	
Ordenador portátil	599€
Licencia Office Home&Student	119€
Conexión ADSL	450€
TOTAL	1168€
COSTE TOTAL	**3043€**

5.3 Entorno socioeconómico

El trabajo no está dirigido a ningún entorno socioeconómico en particular. En España, aproximadamente un 70% de los hogares dispone de conexión a internet [27], por lo que la barrera de acceso más importante está superada.

La importancia de este proyecto reside en eso, en que no responde a una necesidad específica de un grupo de población, sino que ataca un problema que afecta a la mayoría. Hay que destacar que, en el 2012, las llamadas por Skype aumentaron un 5%, llegando a los 490 millones [28], por lo que las llamadas VoIP cada vez están alcanzando una mayor cota de mercado, sobre todo para llamadas internacionales.

Por tanto, dada la proliferación de este tipo de comunicaciones, tanto en redes fijas como en móviles, con aplicaciones como Viber, que ya alcanza los más de 200 millones de usuarios, y otras plataformas de mensajería instantánea que están introduciendo también las llamadas entre sus servicios, es muy importante trabajar para encontrar una solución que mejore la QoE recibida.

Dado que se trata de un proyecto europeo, cualquiera puede pedir una Whitebox rellenando un formulario, y si consideran que las características de tu red, la zona donde se vaya a instalar y otras características del usuario se adecúan a una parte del proyecto que aún no está suficientemente cubierta, te lo envían gratuitamente. Por tanto, puede ocurrir que tras

solicitarlo no seleccionen ese hogar, pero no será por una cuestión socioeconómica, sino por tratar de hacer el estudio lo más amplio posible y recoger datos del mayor número de lugares.

Por tanto, se podría decir que es un proyecto que llega a todas las capas de la sociedad, tanto a particulares como empresas, sin unas grandes exigencias económicas, en un campo que cada vez tiene más usuarios y genera un mayor tráfico en la red, por lo cualquier mejora introducida puede ser bienvenida por un gran número de personas y contribuir al crecimiento de las llamadas y videollamadas de VoIP.

Capítulo 6

Conclusiones

6.1 Conclusiones

Tras completar el desarrollo del proyecto, podemos concluir que:

- Aunque los programas dedicados a las comunicaciones multimedia han mejorado sus prestaciones en los últimos años, su calidad, aún en condiciones óptimas, sigue sin ser excelente.
- Cuando el ancho de banda no es el óptimo, esta falta de calidad se acentúa debido a las pérdidas de paquetes que se producen, debido a que no llegan o llegan tarde y se descartan.
- Las llamadas de VoIP están creciendo mucho en los últimos años, por lo que encontrar una solución que mejore su calidad.
- El hecho de que un códec tenga un bitrate de salida menor a otro no implica que su calidad sea inferior; puede tratarse de una mejor codificación. Por tanto, la selección de un códec que requiera un menor ancho de banda no significa una pérdida de calidad.
- Ciertos codecs no se pudieron incluir en el proyecto debido a que el número que se corresponde con su formato se asigna de forma dinámica, por lo que no podremos compararlos con los que serán válidos con el ancho de banda disponible.

- El programa funciona correctamente cuando se envía audio o vídeo en la solicitud INVITE, seleccionando únicamente los codecs que nos proporcionarán una conexión fluida.
- En el caso de incluirse información multimedia tanto de vídeo como de audio, el problema fallará al procesar el paquete, por lo que no gestiona correctamente los codecs. Pese a ello, en el programa se valora la posibilidad de que ambos tipos de información estén disponibles simultáneamente, y como seleccionar los codecs tanto de audio como de vídeo en ese caso. Además, en la mayoría de los formatos de vídeo, el audio va embebido junto a éste, por lo que no sería necesario un canal específico para ello.
- Cuando se envían numerosas solicitudes seguidas, se dejan de recibir en la pila del programa, y por tanto de gestionarse y reenviarse hacia el servidor. El problema no reside en el tamaño de la pila, pues no tiene ningún tamaño fijo asignado. Por otra parte, no es algo habitual que un cliente envíe tantas solicitudes seguidas al servidor.

En resumen, el sector de las comunicaciones multimedia está creciendo mucho en los últimos años, y su calidad sigue sin ser la mejor. Por tanto, es importante la solución propuesta en este proyecto, pues a pesar de tener algunos fallos que con el tiempo se podrían solucionar, aporta una mejora respecto al estado actual.

6.2 Posibles líneas de trabajo futuro

Existen varias líneas en las que seguir trabajando tras este proyecto. Algunas son más viables que otras, tanto por su complejidad como por la infraestructura necesaria para llevarlas a cabo. Estas soluciones podrían implementarse individualmente o en conjunto. A continuación daremos unas ideas sobre posibles proyectos futuros:

6.2.1 Actualización en tiempo real

Ahora mismo, el programa hace un análisis de las condiciones de la red al negociarse la sesión SIP, pero una vez establecida, no se tiene en cuenta el estado en el que se encuentra la comunicación multimedia, salvo que el cliente envíe un re-INVITE automáticamente.

Una de las posibilidades sería seguir analizando las características de la red en tiempo real, de manera que si éstas cambiasen, se renegociría la sesión para adaptarla a las nuevas necesidades, ya sean mejores o peores.

Además, se podrían analizar más parámetros además del ancho de banda, pues al haber intercambio de mensajes RTCP, podríamos tener en cuenta parámetros como el jitter o la pérdida de paquetes que esté habiendo para obtener un algoritmo mejor y una calidad superior.

Dado que los cambios de ancho de banda pueden ser continuos y existir picos puntuales, probablemente la mejor opción sería trabajar con la capacidad media en un tiempo

"x", en vez de trabajar directamente con el último valor, que puede representar un pico que no se corresponda con la realidad, y por consiguiente, representar una mala decisión.

En resumen, con esta mejora conseguiríamos obtener una situación más real de la evolución de las características de la red con el tiempo, lo que nos permitiría una correcta adaptación de la sesión a ella, y por tanto una experiencia de usuario superior.

6.2.2 Análisis en cliente y servidor(es)

Actualmente, solo el cliente dispone del hardware específico necesario para realizar el análisis del ancho de banda, y por tanto para tomar las decisiones. Esto significa que en la negociación, se tienen en cuenta únicamente las características de la red de acceso del cliente, pero no en el otro extremo.

En cambio, no siempre el cliente es el que representa un "cuello de botella" en la red, también puede existir en el servidor. Si cada usuario final dispusiese de una Whitebox, se podrían tener en cuenta las capacidades de la red en todos los accesos, y negociar la sesión con toda la información disponible.

De esta manera, si en el cliente se detecta un ancho de banda suficiente para los codecs disponibles y no se modifica, pero al enviar la respuesta en el servidor detectamos que no es óptima, se puede modificar antes de enviar la respuesta e iniciar la sesión con otro formato que se adapte mejor a las necesidades. Esto se podría extrapolar también a escenarios más amplios, como una multiconferencia, pues si cada uno de los participantes dispone del equipo necesario, se podrá utilizar la mejor opción para todos, ya sea utilizando la menor calidad para todos, o mediante el uso de traductores y mezcladores para adaptarlo a las necesidades de cada uno.

Por último, si cada usuario dispusiese de una Whitebox, estas podrían "comunicarse" entre ellas y de esta manera obtener más información, como distancias, pérdidas, retardos… que nos permitiesen mejorar la elección, basándola en más criterios además del ancho de banda en acceso, para poder conseguir una mejor comunicación multimedia.

6.2.3 Integración en el router

Para la utilización del programa, es necesario disponer de una Whitebox que proporcione la información necesaria y que ésta esté conectada a nuestro router, lo que nos obliga a añadir un equipo más a nuestra red, y encontrar un sitio donde colocarlo.

Aunque no es básico, sería más cómodo y práctico integrar las características de este equipo directamente en nuestro router, de manera que fuese más sencillo de utilizar para el usuario "normal", que simplemente recibiría una mejor experiencia de usuario en sus comunicaciones multimedia, sin necesidad de comprar ni colocar ningún hardware específico.

Referencias

[1] IETF RFC 3261, *SIP: Session Initiation Protocol*, 2002. Disponible [Internet]: <http://tools.ietf.org/html/rfc3261>[Junio 2002]

[2] SamKnows, 2013. Disponible [Internet]: <http://www.samknows.com/broadband/>

[3] IETF RFC 4566, *SDP: Session Description Protocol*, 2006. Disponible [Internet]: <http://tools.ietf.org/html/rfc4566> [Julio 2006]

[4] CTTC, Marc Cardenete-Suriol, Josep Mangues-Bafalluy, Álvaro Masó, Mónica Gorricho, *Characterization and comparison of Skype behavior in wired and wireless network scenarios*, 2007. Disponible [Internet]: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.1182&rep=rep1&type=pdf>

[5] Rozalina Dimova, Georgi Georgiev, Zlatko Stanchev, *Performance Analysis of QoS Parameters for Voice over IP Applications in a LAN Segment*. 2008. Disponible [Internet]: <http://csconf.org/Volume1/page232.pdf>

[6] Te-Yuan Huang, Polly Huang, Kuan-Ta Chen, Po-Jung Wang, *Could Skype be More Satisfying? A QoE-Centric Study of the FEC Mechanism in an Internet-Scale VoIP System*, Disponible [Internet]:< http://mmnet.iis.sinica.edu.tw/pub/huang10_skype_fec.pdf>

[7] IETF RFC 3550, *RTP: A Transport Protocol for Real-Time Applications*, 2003. Disponible [Internet]: <http://tools.ietf.org/html/rfc3550> [Julio 2003]

[8] IETF RFC 3551, *RTP Profile for Audio and Video Conferences with Minimal Control*, 2003. Disponible [Internet]: <http://tools.ietf.org/html/rfc3551> [Julio 2003]

[9] Netfilter, *iptables project*. Disponible [Internet]: <http://www.netfilter.org/projects/iptables/ >

[10] Pello Xabier Altadill Izura, *iptables: manual práctico*, Disponible, [Internet]: <http://www.pello.info/filez/firewall/iptables.html>

[11] IETF RFC 2543, *SIP: Session Initiation Protocol*, 1999. Disponible [Internet]: <http://tools.ietf.org/html/rfc2543>[Marzo 1999] Obsoleto

[12] IETF RFC 3264, *An Offer/Answer Model with the Session Description Protocol (SDP)*, 2002. Disponible [Internet]: <http://tools.ietf.org/html/rfc3261>[Junio 2002]

[13] IETF RFC 3550, *RTP: A Transport Protocol for Real-Time Applications*, 1996. Disponible [Internet]: <http://tools.ietf.org/html/rfc3550> [Enero 1996] Obsoleto

[14] Stardtot Technologies, *Bandwidth and Storage Calculator*. 2013. Disponible [Internet]: <http://stardot.com/bandwidth-and-storage-calculator>

[15] Khun SA, *CAMARA IP SIP H.264/MJPEG VGA GXV-3615 GRANDSTREAM* . Disponible [Internet]: <http://www.kuhn.cl/webstore/camara-ip-sip-h-264-mjpeg-vga-gxv-3615-grandstream.html>

[16] LBNL's Network Research Group, *H.261/nv Compression Performance*. Disponible [Internet]: < http://ee.lbl.gov/vic/rd.html>

[17] ITU-T, International Telecommunication Union, *H261 Recommendation, VIDEO CODEC FOR AUDIOVISUAL SERVICES AT p* 64 kbits*, 1994. Disponible descarga [Internet]: < http://www.itu.int/rec/T-REC-H.261-199303-I/en>
[18] ISO/IEC 11172-2:1993, *Information technology- Coding moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*, 1993. Disponible previsualización [Internet]: < http://webstore.iec.ch/preview/info_isoiec11172-2%7Bed1.0%7Den.pdf> Versión completa comprar [Internet]: <http://webstore.iec.ch/preview/info_isoiec11172-2%7Bed1.0%7Den.pdf> [1 de agosto 1993]
[19] ITU-T, International Telecommunication Union, *H263 Recommendation, Video coding for low bit rate communication*, 2005. Disponible descarga [Internet]: <
[20] IETF RFC 768, *User Datagram Protocol*, 1980. Disponible, [Internet]: <http://tools.ietf.org/html/rfc768>[Agosto 1980]
[21] IETF RFC 793, *Transmission Control Protocol*, 1981. Disponible, [Internet]: <http://tools.ietf.org/html/rfc793>[Septiembre 1981]
[22] Oracle, *Make the future Java*. Disponible [Internet]: <http://www.oracle.com/es/technologies/java/overview/index.html>
[23] Miguel Ángel Álvarez, Desarrollo Web, *¿Qué es Java?*, 2001. Disponible [Internet]: <http://www.desarrolloweb.com/articulos/497.php> [18 de julio de 2001]
[24] Java.net, The Source for Java Technology Collaboration, *JAIN-SIP project*, 2010. Disponible [Internet]: <https://jsip.java.net/> [Diciembre 2010]
[25] FCC, U.S. Federal Communications Commission, *2012 FCC BROADBAND TESTING AND MEASUREMENT PROGRAM CODE OF CONDUC*T.. Disponible [Internet]: <http://www.samknows.com/broadband/uploads/Code_of_Conduct_v2.0-Feb_14-Signature_Draft.pdf>[17 de febrero de 2012]
[26] Sourceforge SIPp, 2004. Disponible [Internet]: <http://sipp.sourceforge.net/>
[27] Instituto Nacional de Estadística (INE), *Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación (TIC) en los Hogares*, 2012. Disponible [Intenrnet]: < http://www.ine.es/prensa/np738.pdf> [3 de octubre de 2012]
[28] Teresa A., Universo Digital Noticias, *VoIP, Skype gana la tercera parte de las llamadas telefónicas internacionales*, 2013. Disponible [Internet]: <http://universodigitalnoticias.com/internet/18/02/2013/voip-skype-gana-la-tercera-parte-de-las-llamadas-telefonicas-internacionales/6882.html> [18 de febrero de 2013]
[29] Viber. Disponible [Internet]: < http://www.viber.com/about>